PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-099428
(43)Date of publication of application : 05.04.2002

(51)Int.Cl. G06F 9/45

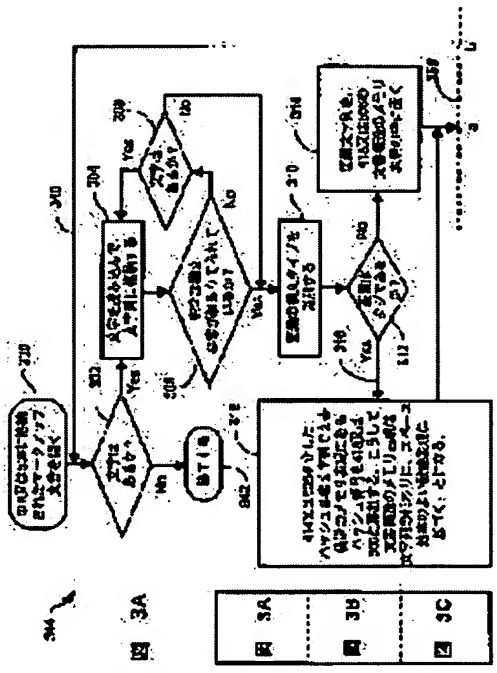
(21)Application number : 2001-201414
(22)Date of filing : 02.07.2001

(71)Applicant : CANON INC
(72)Inventor : JOHN CHARLES BROOKE

(30)Priority
Priority number : 2000 PQ8495 Priority date : 30.06.2000 Priority country : AU

(54) HASH COMPACT XML PARSER

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a method of parsing a markup language document including syntactic elements to improve a competing with a memory and a processing burden in an apparatus having hardware constrains.
SOLUTION: A hash compact XML parser comprises a step 310 of identifying a type of an element for one of the syntactic elements, a step 318 of processing the element by determining a hash representation thereof if the type is a first type and a step 314 of augmenting an at least partial structural representation of the document using the hash representation if the type is the first type.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号
特開2002－99428
（P2002－99428A）
(43)公開日　平成14年４月５日(2002.4.5)

(51)Int.Cl.'	識別記号	ＦＩ	テーマコード'(参考)
Ｇ０６Ｆ　9/45		Ｇ０６Ｆ　9/44	３２２Ｂ　５Ｂ０８１
			３２２Ｃ

審査請求　未請求　請求項の数58　ＯＬ　外国語出願　（全100頁）

(21)出願番号　特願2001－201414(P2001－201414)

(22)出願日　平成13年７月２日(2001.7.2)

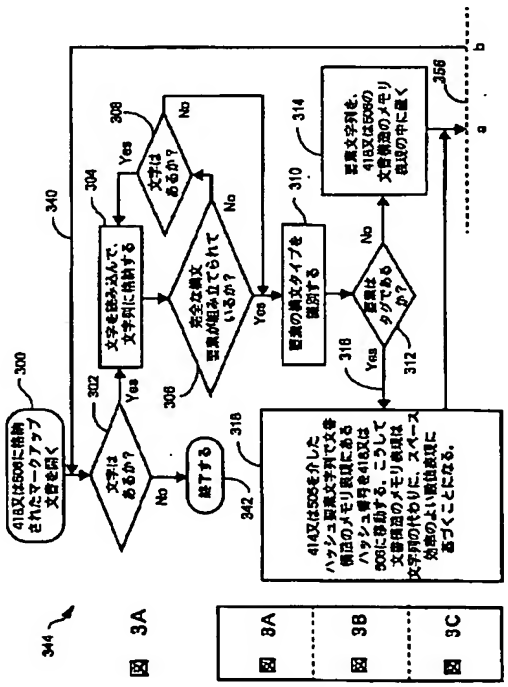
(31)優先権主張番号　ＰＱ８４９５
(32)優先日　平成12年６月30日(2000.6.30)
(33)優先権主張国　オーストラリア（ＡＵ）

(71)出願人　000001007
キヤノン株式会社
東京都大田区下丸子３丁目30番２号
(72)発明者　ジョン　チャールズ　ブルック
オーストラリア国　2113　ニュー　サウス　ウェールズ州、　ノース　ライド、　トーマス　ホルト　ドライブ　１　キヤノン　インフォメーション　システムズ　リサーチ　オーストラリア　プロプライエタリー　リミテツド　内
(74)代理人　100076428
弁理士　大塚　康徳　（外３名）

Ｆターム(参考)　5B081 AA10 CC11 CC61

(54)【発明の名称】　ハッシュコンパクトＸＭＬパーサ

(57)【要約】　（修正有）
【課題】ハードウェア制約のある装置でのメモリとの競合および処理負担を改善する構文要素を含むマークアップ言語文書の解析方法を提供する。
【解決手段】構文要素の１つに対し、その要素のタイプを識別し３１０、前記タイプが第１タイプの場合はハッシュ表現を決定することによってその要素を処理し３１８、前記タイプが前記第１タイプの場合は、文書の少なくとも部分的な構造表現を強める３１４、という手続きを含む。



【特許請求の範囲】
【請求項１】構文要素を含むマークアップ言語文書を解析する解析方法であって、前記構文要素の内の一要素に対し、
前記一要素のタイプを識別する識別ステップと、
前記タイプが第１タイプであれば、そのハッシュ表現を決定することによってその要素を処理する処理ステップと、
前記タイプが前記第１タイプであれば、前記ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、
を含むことを特徴とする解析方法。
【請求項２】イベント駆動型であることを特徴とする請求項１に記載の解析方法。
【請求項３】ハッシュアルゴリズム、
関連ユニバーサルリファレンスインディケータに依存する前記ハッシュアルゴリズムへの第１リファレンス、
関連名前空間に依存する前記ハッシュアルゴリズムへの第２リファレンス、
及び、関連エクステンドマークアップ言語宣言に依存する前記ハッシュアルゴリズムへの第３リファレンス、
のいずれかを用いて前記ハッシュ表現を決定することを特徴とする請求項１に記載の解析方法。
【請求項４】前記第１タイプは、
構造要素或はその一部、
前記構造要素の定義、
前記構造要素の宣言、及び前記構造要素のマッチ、
のいずれかであることを特徴とする請求項２に記載の解析方法。
【請求項５】前記構造要素はタグであることを特徴とする請求項４に記載の解析方法。
【請求項６】前記ハッシュ表現は、前記一つの構文要素ごとのユニークなコードであり、前記一構文要素は第１文字数以下であることを特徴とする請求項２に記載の解析方法。
【請求項７】前記ハッシュ表現は、前記一つの構文要素ごとにユニークなコードではなく、前記一構文要素は、（i）前記要素の文字数、及び(ii)前記要素の文字の順列の許容数、のうちの少なくとも一つの観点から、ある可能性レベルまで制約を受けることを特徴とする請求項２に記載の解析方法。
【請求項８】前記コードは、英数字を含むことを特徴とする請求項６に記載の解析方法。
【請求項９】前記処理ステップは、（ｉ）前記第１タイプの第１インスタンスである前記一構文要素と（ｉｉ）前記第１タイプの第２インスタンスである他方の構文要素と、の両方の拡張ハッシュ表現を決定するサブステップを含み、
前記第１インスタンス内に、前記第２インスタンスがネストになっていることを特徴とする請求項２に記載の解析方法。
【請求項１０】前記構文要素の内の他要素について、更に、
前記他要素のタイプを識別するステップと、
前記他要素のタイプが前記第１タイプと同等である場合に、
（ｉ）前記他要素を処理し、それによってその第２ハッシュ表現を決定する第２処理ステップと、
（ｉｉ）前記第２ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、を有し、
前記処理ステップ及び前記第２処理ステップは、前記一要素と前記他要素の間に第１の関係が存在する場合に、第１の関係に対応する第２の関係が、前記一要素のハッシュ表現と、前記他要素のハッシュ表現との間に存在することを特徴とする請求項１に記載の解析方法。
【請求項１１】前記一要素は、開始タグであり、
前記他要素は、終了タグであり、
前記一要素のハッシュ表現は、対応ハッシュ開始タグであり、
前記他要素の第２ハッシュ表現は、対応ハッシュ終了タグであることを特徴とする請求項１０に記載の解析方法。
【請求項１２】前記終了タグは、前記開始タグの第１変形であり、
前記ハッシュ終了タグは、前記ハッシュ開始タグの第２変形であり、
前記第２変形は、前記第１変形に対応することを特徴とする請求項１１に記載の解析方法。
【請求項１３】前記終了タグは、そこに識別文字が組込まれること以外は、前記開始タグと同一であり、
前記ハッシュ終了タグは、
前記ハッシュ開始タグと同一のもの、
そこに識別文字が組込まれること以外は前記ハッシュ開始タグと同一のもの、及び
オペレータによって処理されたハッシュ開始タグと同一のもの、
の内の少なくとの一つであることを特徴とする請求項１２に記載の解析方法。
【請求項１４】前記一要素及び前記他要素は、それぞれ、開始タグ及び終了タグを含み、第１タグペアをなし、
前記第１タグペアの為の対応ハッシュ開始タグ及び対応ハッシュ終了タグは、前記文書の前記部分的構造表現に組込まれ、
前記文書は、更に、開始タグと終了タグをそれぞれ含む第２タグペアを含み、該第２タグペアは、前記文書の前記第１タグペア内にネストされ、
前記解析方法は、更に、
前記第２タグペアを処理し、対応第２ハッシュ開始／終

了タグを形成するステップと、
前記第２ハッシュ開始／終了タグが、前記第１タグペアの為の前記ハッシュ開始／終了タグとの関係で、前記第１タグペア内での前記第２タグペアのネストと同等のネストを示すため、前記対応第２ハッシュ開始／終了タグを用いて、前記文書の前記少なくとも部分的構造表現を増大する増大ステップと、
を有することを特徴とする請求項１２に記載の解析方法。
【請求項１５】前記増大ステップの前に、更に、
第１ハッシュ開始タグを第２ハッシュ開始タグと連結し、第１ハッシュ終了タグを第２ハッシュ終了タグと連結し、それにより、前記第２タグペアについてのそれぞれの拡張ハッシュ開始／終了タグを形成し、
前記増大ステップは、前記第２タグペアについての前記それぞれの拡張ハッシュ開始／終了タグを用いて実行され、
前記拡張ハッシュ開始／終了タグは、前記第１タグペアの前記ハッシュ開始／終了タグとの関係で、前記第１タグペア内での前記第２タグペアのネストと同等のネストを示すことを特徴とする請求項１４に記載の解析方法。
【請求項１６】前記増大ステップに続いて、構文規則に対する整形性チェックステップが行われ、
該整形性チェックは、前記マークアップ言語文書の前記少なくとも部分的な構造表現における要素の対応ハッシュ表現を数的に比較することによって、前記マークアップ言語文書の前記少なくとも部分的な構造表現を構文規則に照らしてチェックするステップを含むことを特徴とする請求項１に記載の解析方法。
【請求項１７】前記数的な比較のステップに続いて、更に、
前記構文規則に従い、前記第１タイプではない要素の対応非ハッシュ表現を、文字列比較するステップを行うことを特徴とする請求項１６に記載の解析方法。
【請求項１８】前記第１タイプは、
構造要素或はその一部、
前記構造要素の定義、
前記構造要素の宣言、及び前記構造要素のマッチ、
のいずれかであることを特徴とする請求項１に記載の解析方法。
【請求項１９】前記構造要素はタグであることを特徴とする請求項１８に記載の解析方法。
【請求項２０】前記文書の前記少なくとも部分的な構造表現の整形性を、構文規則に照らしてチェックするチェックステップを更に含むことを特徴とする請求項１４に記載の解析方法。
【請求項２１】前記構文規則は、タグの適正なネストに関するものであり、
前記チェックステップは、
前記文書の前記少なくとも部分的な構造表現におけるハッシュタグに亘る数的比較を行い、それにより、前記第１ハッシュ開始／終了タグと、前記第２ハッシュ開始／終了タグを識別するサブステップと、
前記第２ハッシュ開始／終了タグが前記第１ハッシュ開始／終了タグとの関係で適正なネストを示すことを確認するサブステップと、
を含むことを特徴とする請求項２０に記載の解析方法。
【請求項２２】前記数的比較に続いて、
前記構文規則に従い、前記文書の前記少なくとも部分的な構造表現におけるそれぞれのタグの非ハッシュ部分に亘り、文字列比較を行うステップを更に含むことを特徴とする請求項２１に記載の解析方法。
【請求項２３】構文規則に照らして、前記文書の前記少なくとも部分的な構文表現の整形性をチェックするステップを更に有することを特徴とする請求項１５に記載の解析方法。
【請求項２４】前記構文規則は、タグの適正なネストに関するものであり、
前記チェックステップは、
前記文書の前記少なくとも部分的な構造表現におけるハッシュタグに亘り数的比較を行い、それにより、前記第１ハッシュ開始／終了タグと、前記第２ハッシュ開始／終了タグを識別するサブステップと、
前記拡張ハッシュ開始／終了タグが、前記第１ハッシュ開始／終了タグとの関係で適正なネストを示すことを確認するサブステップと、
を含むことを特徴とする請求項２３に記載の解析方法。
【請求項２５】前記数的比較に続いて、
前記文書の前記少なくとも部分的な構造表現におけるそれぞれのタグの非ハッシュ部分にわたり、文字列比較を行うステップを更に含むことを特徴とする請求項２４に記載の解析方法。
【請求項２６】前記整形性チェックステップは、妥当性参照文書ＶＲＤに照らす妥当性検査ステップによって、
（ａ）続けられ、（ｂ）含まれ、又は（ｃ）置き換えられ、
前記妥当性検査ステップは、
（ａ）前記ＶＲＤを処理する処理サブステップを含み、該処理サブステップは、ＶＲＤにおけるの構文要素ごとに、
（ｉ）ＶＲＤの前記構文要素のタイプを識別するサブサブステップと、
（ｉｉ）前記タイプが、前記第１タイプの場合に、前記構文要素を、そのハッシュ表現を決定することによって処理するサブサブステップと、を含み、
前記妥当性検査ステップは、更に、
（ｂ）処理されたＶＲＤに照らして前記マークアップ言語文書の前記少なくとも部分的な構造表現をチェックするチェックサブステップを含み、該チェックサブステップは、要素の対応ハッシュ表現を数的に比較するサブサ

ブステップを含むことを特徴とする請求項１６に記載の解析方法。
【請求項２７】ＶＲＤに照らしてマークアップ言語文書の妥当性検査を行う検査方法であって、
（ａ）マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第１ドキュメントタグでなければ、処理するステップを有し、該処理は、
（ｉ）前記ドキュメントタグの階層位置を決定するステップと、
（ｉｉ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するステップと、
（ｉｉｉ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するステップと、を含み、
前記検査方法は更に、
（ｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理するステップを有し、該処理は、
（ｉ）前記タグの階層位置を決定するステップと、
（ｉｉ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するステップと、
（ｉｉｉ）前記タグの前記拡張ハッシュ表現をリストに格納するステップと、を含み、
前記検査方法は更に、
（ｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合には、前記マークアップ言語文書の妥当性を検査するステップを有することを特徴とする検査方法。
【請求項２８】ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する検査方法であって、
（ａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理する処理ステップを含み、該処理ステップは、
（ｉ）前記構造要素の構文属性を決定するステップと、
（ｉｉ）前記構造要素のハッシュ表現を決定するステップと、
（ｉｉｉ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するステップと、を含み、
前記検査方法は更に、
（ｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理する処理ステップを含み、該処理ステップは、
（ｉ）前記文書構造要素の構文属性を決定するステップと、
（ｉｉ）前記文書構造要素のハッシュ表現を決定するステップと、
（ｉｉｉ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するステップと、を含み、
前記検査方法は更に、
（ｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するステップを有することを特徴とする検査方法。
【請求項２９】前記数的比較ステップに続いて、更に、前記第１タイプでない要素の対応非ハッシュ表現の文字列比較ステップを行うことを特徴とする請求項２６に記載の解析方法。
【請求項３０】前記第１タイプは、
構造要素或はその一部、
前記構造要素の定義、
前記構造要素の宣言、
前記構造要素のマッチ、
のいずれかであることを特徴とする請求項２６に記載の解析方法。
【請求項３１】前記構造要素はタグであることを特徴とする請求項３０に記載の解析方法。
【請求項３２】構文要素を含むマークアップ言語文書を符号化する符号化方法であって、
前記構文要素の一つについて、
構文要素のタイプを識別するステップと、
（ｉ）前記タイプが第１タイプであれば、前記構文要素のハッシュ表現を決定すること、
（ｉｉ）前記タイプが第１タイプでなければ、前記構文要素の圧縮表現を決定すること、
（ｉｉｉ）前記構文要素を保持すること、
のいずれかによって、前記構文要素を処理するステップと、
を含むことを特徴とする符号化方法。
【請求項３３】符号化構文要素を含むマークアップ言語文書を復号する復号方法であって、
前記符号化構文要素の一つについて、
前記符号化構文要素のタイプを識別するステップと、
（ｉ）前記タイプが第１タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、
（ｉｉ）前記タイプが第１タイプでなければ、前記符号化構文要素の解凍表現を決定すること、
（ｉｉｉ）前記符号化構文要素を保持すること、
の少なくとも一つで、前記符号化構文要素を処理するステップと、
を含むことを特徴とする復号方法。
【請求項３４】構文要素を含むマークアップ言語文書を

解析する解析装置であって、
前記構文要素のタイプを識別する識別手段と、
前記タイプが第１タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理する処理手段と、
前記タイプが第１タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大する増大手段と、
を含むことを特徴とする解析装置。
【請求項３５】ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、
（ａ）マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第１ドキュメントタグでなければ、処理する手段を有し、該手段は、
（ｉ）前記ドキュメントタグの階層位置を決定する手段と、
（ｉｉ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定する手段と、
（ｉｉｉ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納する手段と、を含み、
前記検査装置は更に、
（ｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理する手段を有し、該手段は、
（ｉ）前記タグの階層位置を決定する手段と、
（ｉｉ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定する手段と、
（ｉｉｉ）前記タグの前記拡張ハッシュ表現をリストに格納する手段と、を含み、
前記検査装置は更に、
（ｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットであるかのどちらかであることを立証し、それによって、前記マークアップ言語文書の妥当性を検査する手段を有することを特徴とする検査装置。
【請求項３６】ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、
（ａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理する手段を含み、該手段は、
（ｉ）前記構造要素の構文属性を決定する手段と、
（ｉｉ）前記構造要素のハッシュ表現を決定する手段と、
（ｉｉｉ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納する手段と、を含み、
前記検査装置は更に、
（ｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理する手段を含み、該手段は、
（ｉ）前記文書構造要素の構文属性を決定する手段と、
（ｉｉ）前記文書構造要素のハッシュ表現を決定する手段と、
（ｉｉｉ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納する手段と、を含み、
前記検査装置は更に、
（ｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現と、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現とを比較し、それにより、前記マークアップ言語文書の妥当性を検査する手段を有することを特徴とする検査装置。
【請求項３７】構文要素を含むマークアップ言語文書を符号化し、前記文書の少なくとも部分的構造表現を形成する符号化装置であって、
構文要素のタイプを識別する手段と、
（ｉ）前記タイプが第１タイプであれば、前記構文要素のハッシュ表現を決定すること、
（ｉｉ）前記タイプが第１タイプでなければ、前記構文要素の圧縮表現を決定すること、
（ｉｉｉ）前記構文要素を保持すること、
のいずれかによって、前記構文要素を処理する手段と、
を含むことを特徴とする符号化装置。
【請求項３８】符号化構文要素を含むマークアップ言語文書を復号する復号装置であって、
前記符号化構文要素のタイプを識別する手段と、
（ｉ）前記タイプが第１タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、
（ｉｉ）前記タイプが第１タイプでなければ、前記符号化構文要素の解凍表現を決定すること、
（ｉｉｉ）符号化構文要素を保持すること、
の少なくとも一つによって、前記符号化構文要素を処理する手段と、
を含むことを特徴とする復号装置。
【請求項３９】コンピュータに構文要素を含むマークアップ言語文書を解析する手続を実行させるコンピュータプログラムであって、
前記構文要素のタイプを識別するコードと、
前記タイプが第１タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、
前記タイプが第１タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、
を含むことを特徴とするコンピュータプログラム。

【請求項４０】コンピュータにＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムであって、
（ａ）マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第１ドキュメントタグでなければ、処理するコードを有し、該コードは、
（ｉ）前記ドキュメントタグの階層位置を決定するコードと、
（ｉｉ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、
（ｉｉｉ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、
前記コンピュータプログラムは更に、
（ｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理するコードを有し、該コードは、
（ｉ）前記タグの階層位置を決定するコードと、
（ｉｉ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、
（ｉｉｉ）前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、
前記コンピュータプログラムは更に、
（ｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合には、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム。
【請求項４１】コンピュータにＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムであって、
（ａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、
（ｉ）前記構造要素の構文属性を決定するコードと、
（ｉｉ）前記構造要素のハッシュ表現を決定するコードと、
（ｉｉｉ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、
前記プログラムは更に、
（ｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、
（ｉ）前記文書構造要素の構文属性を決定するコードと、
（ｉｉ）前記文書構造要素のハッシュ表現を決定するコードと、
（ｉｉｉ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、
前記プログラムは更に、
（ｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム。
【請求項４２】コンピュータに構文要素を含むマークアップ言語文書を符号化する手続を実行させるコンピュータプログラムであって、
構文要素のタイプを識別するコードと、
（ｉ）前記タイプが第１タイプであれば、前記構文要素のハッシュ表現を決定すること、
（ｉｉ）前記タイプが第１タイプでなければ、前記構文要素の圧縮表現を決定すること、
（ｉｉｉ）前記構文要素を保持すること、
のいずれかによって、前記構文要素を処理するコードと、
を含むことを特徴とするコンピュータプログラム。
【請求項４３】コンピュータに符号化構文要素を含むマークアップ言語文書を復号する手続を実行させるコンピュータプログラムであって、
前記符号化構文要素のタイプを識別するコードと、
（ｉ）前記タイプが第１タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、
（ｉｉ）前記タイプが第１タイプでなければ、前記符号化構文要素の解凍表現を決定すること、
（ｉｉｉ）符号化構文要素を保持すること、
の少なくとも一つで、前記符号化構文要素を処理するコードと、
を含むことを特徴とするコンピュータプログラム。
【請求項４４】コンピュータに構文要素を含むマークアップ言語文書を解析する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、
前記コンピュータプログラムは、
前記構文要素のタイプを識別するコードと、
前記タイプが第１タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、
前記タイプが第１タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、
を含むことを特徴とするコンピュータプログラム製品。
【請求項４５】コンピュータにＶＲＤに照らしてマーク

アップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、
前記コンピュータプログラムは、
（ａ）マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第１ドキュメントタグでなければ、処理するコードを有し、該コードは、
（ｉ）前記ドキュメントタグの階層位置を決定するコードと、
（ｉｉ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、
（ｉｉｉ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、
前記コンピュータプログラムは更に、
（ｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理するコードを有し、該コードは、
（ｉ）前記タグの階層位置を決定するコードと、
（ｉｉ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、
（ｉｉｉ）前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、
前記コンピュータプログラムは更に、
（ｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム製品。
【請求項４６】コンピュータにＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、
前記プログラムは、
（ａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、
（ｉ）前記構造要素の構文属性を決定するコードと、
（ｉｉ）前記構造要素のハッシュ表現を決定するコードと、
（ｉｉｉ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、
前記プログラムは更に、
（ｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、
（ｉ）前記文書構造要素の構文属性を決定するコードと、
（ｉｉ）前記文書構造要素のハッシュ表現を決定するコードと、
（ｉｉｉ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、
前記プログラムは更に、
（ｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム製品。
【請求項４７】構文要素を含むマークアップ言語文書の少なくとも部分的な構造表現であって、
前記構文要素の内の一要素に対し、
前記一要素のタイプを識別する識別ステップと、
前記タイプが第１タイプであれば、そのハッシュ表現を決定することによってその要素を処理する処理ステップと、
前記タイプが前記第１タイプであれば、前記ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、
を行う解析方法によって生成されることを特徴とする構造表現。
【請求項４８】構文要素を含むマークアップ言語文書を解析する解析装置であって、
プロセッサと、（ｉ）前記文書と、（ｉｉ）前記文書を解析する手続を前記プロセッサに実行させるプログラムを格納したプログラムと、格納するメモリを含み、
該プログラムは、
（ｉ）前記構文要素のタイプを識別するコードと、
（ｉｉ）前記タイプが第１タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、
（ｉｉｉ）前記タイプが第１タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、
を含むことを特徴とする解析装置。
【請求項４９】構文要素を含むＶＲＤに照らして構文要素を含むマークアップ言語文書の妥当性を検査する検査装置であって、
（ａ）プロセッサと、
（ｂ）（ｉ）前記文書と、（ｉｉ）前記ＶＲＤと、（ｉｉｉ）文書の妥当性を検査する手順を前記プロセッサに実行させるプログラムと、を格納するメモリを有し、
（ｃ）前記プログラムは、
（ｃａ）マークアップ言語文書を、そこで識別されるド

キュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第１ドキュメントタグでなければ、処理するコードを有し、該コードは、
（ｃａａ）前記ドキュメントタグの階層位置を決定するコードと、
（ｃａｂ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、
（ｃａｃ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、
前記プログラムは更に、
（ｃｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理するコードを有し、該コードは、
（ｃｂａ）前記タグの階層位置を決定するコードと、
（ｃｂｂ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、
（ｃｂｃ）前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、
前記プログラムは更に、
（ｃｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットであるかのいずれかであることを立証し、それにより前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とする検査装置。
【請求項５０】構文要素を含むＶＲＤに照らして構文要素を含むマークアップ言語文書の妥当性を検査する検査装置であって、
（ａ）プロセッサと、
（ｂ）（ｉ）前記文書と、（ｉｉ）前記ＶＲＤと、（ｉｉｉ）文書の妥当性を検査する手順を前記プロセッサに実行させるプログラムと、を格納するためのメモリを有し、
（ｃ）前記プログラムは、
（ｃａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、
（ｃａａ）前記構造要素の構文属性を決定するコードと、
（ｃａｂ）前記構造要素のハッシュ表現を決定するコードと、
（ｃａｃ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、
前記プログラムは更に、
（ｃｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、
（ｃｂａ）前記文書構造要素の構文属性を決定するコードと、
（ｃｂｂ）前記文書構造要素のハッシュ表現を決定するコードと、
（ｃｂｃ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、
前記プログラムは更に、
（ｃｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現を、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現と比較し、それにより前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とする検査装置。
【請求項５１】ＶＲＤに照らしてマークアップ言語文書の妥当性検査を行う検査方法であって、
前記ＶＲＤにおける第１タイプの最も深くネストされた構文要素についての第１拡張ハッシュ表現を決定するステップと、
ＶＲＤリストに前記第１拡張ハッシュ表現を格納するステップと、
マークアップ言語文書における第１タイプの最も深くネストされた構文要素について、第２拡張ハッシュ表現を決定するステップと、
前記第２拡張ハッシュ表現が前記ＶＲＤリストに存在する場合に、前記マークアップ言語文書は妥当でないことを宣言するステップと、
を含むことを特徴とする検査方法。
【請求項５２】前記第１タイプの前記構文要素は、
構造要素及びその一部、
前記構造要素の定義、
前記構造要素の宣言、
前記構造要素のマッチ、
のいずれかであることを特徴とする請求項５１に記載の検査方法。
【請求項５３】前記構造要素はタグであることを特徴とする請求項５２に記載の検査方法。
【請求項５４】前記ＶＲＤの前記最も深くネストされた構文要素は、前記ＶＲＤの広域的構造、或は前記ＶＲＤの局所的サブ構造のいずれかの中に最も深くネストされた構文要素であり、
前記マークアップ言語文書の最も深くネストされた構文要素は、前記マークアップ言語文書の広域的構造、或は、前記マークアップ言語文書の局所的サブ構造のいずれかの中に最も深くネストされた構文要素であることを特徴とする請求項５１に記載の検査方法。
【請求項５５】ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、
前記ＶＲＤの第１タイプの最も深くネストされた構文要素について第１拡張ハッシュ表現を決定する手段と、

ＶＲＤリストの前記第１拡張ハッシュ表現を格納する手段と、
前記マークアップ言語文書における第１タイプの最も深くネストされた構文要素について、第２拡張ハッシュ表現を決定する手段と、
前記第２拡張ハッシュ表現が前記ＶＲＤリストに存在する場合に、前記マークアップ言語文書は妥当でないと宣言する手段と、
を有することを特徴とする検査装置。
【請求項５６】ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続をコンピュータに実行させるコンピュータプログラムであって、
前記ＶＲＤの第１タイプの最も深くネストされた構文要素について第１拡張ハッシュ表現を決定するコードと、
ＶＲＤリストの前記第１拡張ハッシュ表現を格納するコードと、
前記マークアップ言語文書における第１タイプの最も深くネストされた構文要素について、第２拡張ハッシュ表現を決定するコードと、
前記第２拡張ハッシュ表現が前記ＶＲＤリストに存在する場合に、前記マークアップ言語文書は妥当でないと宣言するコードと、
を含むことを特徴とするコンピュータプログラム。
【請求項５７】ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続をコンピュータに実行させるコンピュータプログラムを格納したコンピュータに含まれるコンピュータプログラム製品であって、該プログラムは、
前記ＶＲＤの第１タイプの最も深くネストされた構文要素について第１拡張ハッシュ表現を決定するコードと、
ＶＲＤリストに前記第１拡張ハッシュ表現を格納するコードと、
前記マークアップ言語文書における前記第１タイプの最も深くネストされた構文要素について第２拡張ハッシュ表現を決定するコードと、
前記第２拡張ハッシュ表現が前記ＶＲＤリストに存在する場合に、前記マークアップ言語文書が妥当でないと宣言するコードと、
を含むことを特徴とするコンピュータプログラム製品。
【請求項５８】前記コードは、英数字を含むことを特徴とする請求項７に記載の解析方法。
【発明の詳細な説明】
【０００１】
【著作権についての注意】この特許明細書は、著作権保護の対象となる素材を含む。その著作権者は、特許庁のファイルから閲覧する目的で、本特許明細書、或は、関連素材を複製することに対しては、なんらの異論もないが、そうでなければ、全てのいかなる著作権もこれを保有する。
【０００２】
【発明の属する技術分野】本発明は、一般的にマルチメディア文書の処理、特に、マークアップ言語で記述された文書に関する。本発明は、マークアップ言語文書を構文解析する方法及び装置に関するものである。本発明はまた、マークアップ言語で記述された文書を構文解析するための処理をコンピュータに実行させるために作成されたコンピュータプログラムに関し、更に、このコンピュータプログラムが記録されたコンピュータ可読媒体を含むコンピュータプログラム製品に関する。
【０００３】
【技術的背景】構文解析（パーシング）とは、文書から情報を抽出する処理である。この処理には、通常、文書の構文に対する最低限のチェックが少なくとも含まれる。そして、一般的に、その文書のツリー構造での記述、又は、イベントの論理的なつながり（logical chain）を得ることができる。そのイベントの論理的なつながりに基づいた構造表現は、一般的に、文頭から文末まで、文書を順番に構文解析することによって生成される。
【０００４】ツリー型のパーサは、例えば、ＸＭＬ文書を内部ツリー構造にコンパイルし、アプリケーションが扱うことのできる階層モデルを与える。Ｗ３Ｃの文書オブジェクトモデル（ＤＯＭ）作業グループは、拡張可能なマークアップ言語（ＸＭＬ）文書に用いる標準ツリー型アプリケーションプログラムインターフェース（ＡＰＩ）を現在開発している。他方、イベント型のパーサは各要素の開始や終了等の解析イベントを、アプリケーション－このアプリケーションのために解析が行われている－に直接報告する。この報告は典型的にはコールバックを用いて実行され、内部ツリー構造を必要としない。解析が必要なアプリケーションは、様々なイベントを処理するためにハンドラを実行する。これはグラフィカルユーザインターフェースにおけるイベントのハンドリングによく似ている。
【０００５】ツリー型のパーサは、広範囲のアプリケーションに有用だが、一般的に、システムリソースに負担がかかり、特に、解析対象となっている文書が大きな場合には負担となる。さらに、アプリケーションは、そのアプリケーション独自のツリー構造の構築を必要とする場合があるが、ツリー表現を、異なる表現と照らし合わせるためだけに、構築するのは非効率である。イベント型のパーサによれば、ＸＭＬ文書に対し、より単純で、より低いレベルでのアクセスが可能となり、使用可能なシステムメモリよりも大きなメモリを要する文書の解析を容易にする。“Ｓｉｍｐｌｅ ＡＰＩ ｆｏｒ ＸＭＬ”（ＳＡＸパーサと称す）は、ＸＭＬ文書を解析するためのイベント駆動型インターフェースである。ＳＡＸパーサは、図２Ａ、図２Ｂ、図３Ａ、図３Ｂ、図３Ｃにおいて、更に詳細に議論する。
【０００６】図１Ａ、図１Ｂは、パーサシステムのブロ

ック図を示したものである。ここでは、以下のＸＭＬ文書部分１０６について考察する。

```
105 <:Shakespeare>:
110 <:!—This is a comment—>:
115 <:div class="preface"Name1="value1"name2="value2">:
120 <:mult list=&:lt:>:<:/mult>:
125 <:banquo>:
130 Say
135 <:quote>:
140 goodnight<:/quote>:,
145 Hamlet.<:/banquo>:
150 <:Hamlet>:<:quote>:Goodnight,Hamlet.<:/quote>:<:/Hamlet>:
155 <:/Shakespeare>:
```

［１］

【０００８】図１Ｂにおいて、ＸＭＬ文書１０６は、この例では、イベント型のパーサである、パーサ１１２へ入力される。点線枠１０８で示されるように、文書型定義（ＤＴＤ）又はＸＭＬスキーマもまた、パーサ１１２へオプションとして入力される。パーサ１１２は、矢印１１４で示されるように、簡単なリストから成る文書１０６の部分的な構造表現を出力する。図１Ａにおいて、カスケードスタイルシート（ＣＳＳ）又は拡張スタイルシート（ＸＳＬ）１０４は、ＣＳＳ又はＸＳＬパーサ１１０へ入力される。ＤＴＤ１０２もまたこのパーサ１１０へ入力される。本図において、ＸＭＬパーサ１１２及びＣＳＳ／ＸＳＬパーサ１１０は共に、イベント駆動型パーサである。

【０００９】ＸＭＬのようなマークアップ言語の利点の１つは、タグを使用して文書の様々な部分を定義できることにより、文書を、容易に、よりスマートで、よりポータブルで、よりパワフルにできることにある。この機能は、ＸＭＬの記述上の性質からきている。ＸＭＬ文書はサブジェクト単位でカスタマイズされる。そして、それにより、カスタマイズされたタグは、人間が文書を構造的に理解できるようにするために用いられる。しかし、この特性によって、ＸＭＬ文書が冗長で大きなものになることがよくあり、場合によっては問題となる。例えば、ＸＭＬ文書がプリンタのように、ハードウェア制約のある装置で解析されなければならない場合、メモリをたくさん使用する性質のある従来型の解析は、そのような装置に格納できる限られたメモリと競合してしまう。さらに、ハードウェア制約のある装置によって文書が処理される場合には、人間のＸＭＬ文書の読みやすさは、一般的に、余り利点にはならない。さらに、ＸＭＬ文書の解析において、タグ文字列のマッチング操作が多く必要とされ、許容できないほどの処理要求の負担がかかり、ひいては、許容できない数のプロセッササイクルを引き起す。これらの問題は、図１Ａ及び図１Ｂに示される両方のパーサ例に当てはまる。

【００１０】

【本発明の開示】本発明の目的は、既存の機器の一つ以上の欠点をほぼ無くし、或は、少なくとも改善することにある。

【００１１】本発明の第１の態様では、構文要素を含むマークアップ言語文書を解析する解析方法であって、前記構文要素の内の一要素に対し、前記一要素のタイプを識別する識別ステップと、前記タイプが第１タイプであれば、そのハッシュ表現を決定することによってその要素を処理する処理ステップと、前記タイプが前記第１タイプであれば、前記ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、を含むことを特徴とする解析方法が提供される。

【００１２】本発明の他の態様では、ＶＲＤに照らしてマークアップ言語文書の妥当性検査を行う検査方法であって、（ａ）マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第１ドキュメントタグでなければ、処理するステップを有し、該処理は、（ｉ）前記ドキュメントタグの階層位置を決定するステップと、（ｉｉ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するステップと、（ｉｉｉ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するステップと、を含み、前記検査方法は更に、（ｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理するステップを有し、該処理は、（ｉ）前記タグの階層位置を決定するステップと、（ｉｉ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するステップと、（ｉｉｉ）前記タグの前記拡張ハッシュ表現をリストに格納するステップと、を含み、前記検査方法は更に、（ｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合には、前記マークアップ言語文書の妥当性を検査するステップを有することを特徴とする検査方法が提

供される。

【００１３】本発明の他の態様では、ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する検査方法であって、（ａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理する処理ステップを含み、該処理ステップは、（ｉ）前記構造要素の構文属性を決定するステップと、（ｉｉ）前記構造要素のハッシュ表現を決定するステップと、（ｉｉｉ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するステップと、を含み、前記検査方法は更に、

（ｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理する処理ステップを含み、該処理ステップは、（ｉ）前記文書構造要素の構文属性を決定するステップと、（ｉｉ）前記文書構造要素のハッシュ表現を決定するステップと、（ｉｉｉ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するステップと、を含み、前記検査方法は更に、（ｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するステップを有することを特徴とする検査方法が提供される。

【００１４】本発明の他の態様では、構文要素を含むマークアップ言語文書を符号化する符号化方法であって、前記構文要素の一つについて、構文要素のタイプを識別するステップと、（ｉ）前記タイプが第１タイプであれば、前記構文要素のハッシュ表現を決定すること、（ｉｉ）前記タイプが第１タイプでなければ、前記構文要素の圧縮表現を決定すること、（ｉｉｉ）前記構文要素を保持すること、のいずれかによって、前記構文要素を処理するステップと、を含むことを特徴とする符号化方法が提供される。

【００１５】本発明の他の態様では、符号化構文要素を含むマークアップ言語文書を復号する復号方法であって、前記符号化構文要素の一つについて、前記符号化構文要素のタイプを識別するステップと、（ｉ）前記タイプが第１タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、（ｉｉ）前記タイプが第１タイプでなければ、前記符号化構文要素の解凍表現を決定すること、（ｉｉｉ）前記符号化構文要素を保持すること、の少なくとも一つで、前記符号化構文要素を処理するステップと、を含むことを特徴とする復号方法が提供される。

【００１６】本発明の他の態様では、構文要素を含むマークアップ言語文書を解析する解析装置であって、前記構文要素のタイプを識別する識別手段と、前記タイプが第１タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理する処理手段と、前記タイプが第１タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大する増大手段と、を含むことを特徴とする解析装置が提供される。

【００１７】本発明の他の態様では、ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、（ａ）マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第１ドキュメントタグでなければ、処理する手段を有し、該手段は、（ｉ）前記ドキュメントタグの階層位置を決定する手段と、（ｉｉ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定する手段と、（ｉｉｉ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納する手段と、を含み、前記検査装置は更に、（ｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理する手段を有し、該手段は、（ｉ）前記タグの階層位置を決定する手段と、（ｉｉ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定する手段と、（ｉｉｉ）前記タグの前記拡張ハッシュ表現をリストに格納する手段と、を含み、前記検査装置は更に、（ｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットであるかのどちらかであることを立証し、それによって、前記マークアップ言語文書の妥当性を検査する手段を有することを特徴とする検査装置が提供される。

【００１８】本発明の他の態様では、ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、（ａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理する手段を含み、該手段は、（ｉ）前記構造要素の構文属性を決定する手段と、（ｉｉ）前記構造要素のハッシュ表現を決定する手段と、（ｉｉｉ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納する手段と、を含み、前記検査装置は更に、（ｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理する手段を含み、該手段は、（ｉ）前記文書構造要素の構文属性を決定する手段と、（ｉｉ）前記文書構造要素のハッシュ表現を決定する手段と、（ｉｉｉ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納する手段と、を含み、前記検査装置は更に、（ｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現と、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現とを比較し、それにより、前記マークアップ言語文書の妥当性を検査する手段を有することを特徴とする検

査装置が提供される。

【００１９】本発明の他の態様では、構文要素を含むマークアップ言語文書を符号化し、前記文書の少なくとも部分的構造表現を形成する符号化装置であって、構文要素のタイプを識別する手段と、（ｉ）前記タイプが第１タイプであれば、前記構文要素のハッシュ表現を決定すること、（ｉｉ）前記タイプが第１タイプでなければ、前記構文要素の圧縮表現を決定すること、（ｉｉｉ）前記構文要素を保持すること、のいずれかによって、前記構文要素を処理する手段と、を含むことを特徴とする符号化装置が提供される。

【００２０】本発明の他の態様では、符号化構文要素を含むマークアップ言語文書を復号する復号装置であって、前記符号化構文要素のタイプを識別する手段と、（ｉ）前記タイプが第１タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、（ｉｉ）前記タイプが第１タイプでなければ、前記符号化構文要素の解凍表現を決定すること、（ｉｉｉ）符号化構文要素を保持すること、の少なくとも一つによって、前記符号化構文要素を処理する手段と、を含むことを特徴とする復号装置が提供される。

【００２１】本発明の他の態様では、コンピュータに構文要素を含むマークアップ言語文書を解析する手続を実行させるコンピュータプログラムであって、前記構文要素のタイプを識別するコードと、前記タイプが第１タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、前記タイプが第１タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、を含むことを特徴とするコンピュータプログラムが提供される。

【００２２】本発明の他の態様では、コンピュータにＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムであって、（ａ）マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第１ドキュメントタグでなければ、処理するコードを有し、該コードは、（ｉ）前記ドキュメントタグの階層位置を決定するコードと、（ｉｉ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、（ｉｉｉ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、前記コンピュータプログラムは更に、（ｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理するコードを有し、該コードは、（ｉ）前記タグの階層位置を決定するコードと、（ｉｉ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、（ｉｉｉ）前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、前記コンピュータプログラムは更に、（ｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合には、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラムが提供される。

【００２３】本発明の他の態様では、コンピュータにＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムであって、（ａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、（ｉ）前記構造要素の構文属性を決定するコードと、（ｉｉ）前記構造要素のハッシュ表現を決定するコードと、（ｉｉｉ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、前記プログラムは更に、（ｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、（ｉ）前記文書構造要素の構文属性を決定するコードと、（ｉｉ）前記文書構造要素のハッシュ表現を決定するコードと、（ｉｉｉ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、前記プログラムは更に、（ｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラムが提供される。

【００２４】本発明の他の態様では、コンピュータに構文要素を含むマークアップ言語文書を符号化する手続を実行させるコンピュータプログラムであって、構文要素のタイプを識別するコードと、（ｉ）前記タイプが第１タイプであれば、前記構文要素のハッシュ表現を決定すること、（ｉｉ）前記タイプが第１タイプでなければ、前記構文要素の圧縮表現を決定すること、（ｉｉｉ）前記構文要素を保持すること、のいずれかによって、前記構文要素を処理するコードと、を含むことを特徴とするコンピュータプログラムが提供される。

【００２５】本発明の他の態様では、コンピュータに符号化構文要素を含むマークアップ言語文書を復号する手続を実行させるコンピュータプログラムであって、前記符号化構文要素のタイプを識別するコードと、（ｉ）前記タイプが第１タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、（ｉｉ）前記タイプが第１タイプでなければ、前記符号化構文要素の解凍表現を決定すること、（ｉｉｉ）符号化構文要素を保持する

こと、の少なくとも一つで、前記符号化構文要素を処理するコードと、を含むことを特徴とするコンピュータプログラムが提供される。

【００２６】本発明の他の態様では、コンピュータに構文要素を含むマークアップ言語文書を解析する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、前記コンピュータプログラムは、前記構文要素のタイプを識別するコードと、前記タイプが第１タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、前記タイプが第１タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、を含むことを特徴とするコンピュータプログラム製品が提供される。

【００２７】本発明の他の態様では、コンピュータにＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、前記コンピュータプログラムは、（ａ）マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第１ドキュメントタグでなければ、処理するコードを有し、該コードは、（ｉ）前記ドキュメントタグの階層位置を決定するコードと、（ｉｉ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、（ｉｉｉ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、前記コンピュータプログラムは更に、

（ｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理するコードを有し、該コードは、（ｉ）前記タグの階層位置を決定するコードと、（ｉｉ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、（ｉｉｉ）前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、前記コンピュータプログラムは更に、（ｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム製品が提供される。

【００２８】本発明の他の態様では、コンピュータにＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、前記プログラムは、（ａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、（ｉ）前記構造要素の構文属性を決定するコードと、（ｉｉ）前記構造要素のハッシュ表現を決定するコードと、（ｉｉｉ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、前記プログラムは更に、（ｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、（ｉ）前記文書構造要素の構文属性を決定するコードと、（ｉｉ）前記文書構造要素のハッシュ表現を決定するコードと、（ｉｉｉ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、前記プログラムは更に、（ｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム製品が提供される。

【００２９】本発明の他の態様では、構文要素を含むマークアップ言語文書の少なくとも部分的な構造表現であって、前記構文要素の内の一要素に対し、前記一要素のタイプを識別する識別ステップと、前記タイプが第１タイプであれば、そのハッシュ表現を決定することによってその要素を処理する処理ステップと、前記タイプが前記第１タイプであれば、前記ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、を行う解析方法によって生成されることを特徴とする構造表現が提供される。

【００３０】本発明の他の態様では、構文要素を含むマークアップ言語文書を解析する解析装置であって、プロセッサと、（ｉ）前記文書と、（ｉｉ）前記文書を解析する手続を前記プロセッサに実行させるプログラムを格納したプログラムと、格納するメモリを含み、該プログラムは、（ｉ）前記構文要素のタイプを識別するコードと、（ｉｉ）前記タイプが第１タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、（ｉｉｉ）前記タイプが第１タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、を含むことを特徴とする解析装置が提供される。

【００３１】本発明の他の態様では、構文要素を含むＶＲＤに照らして構文要素を含むマークアップ言語文書の妥当性を検査する検査装置であって、（ａ）プロセッサと、（ｂ）（ｉ）前記文書と、（ｉｉ）前記ＶＲＤと、（ｉｉｉ）文書の妥当性を検査する手順を前記プロセッサに実行させるプログラムと、を格納するメモリを有し、（ｃ）前記プログラムは、（ｃａ）マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のド

キュメントタグ階層における第１ドキュメントタグでなければ、処理するコードを有し、該コードは、（ｃａａ）前記ドキュメントタグの階層位置を決定するコードと、（ｃａｂ）前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、（ｃａｃ）前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、前記プログラムは更に、（ｃｂ）ＶＲＤを、そこで識別されるタグごとに、該タグが対応タグ階層の第１タグでない場合に、処理するコードを有し、該コードは、（ｃｂａ）前記タグの階層位置を決定するコードと、（ｃｂｂ）対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、（ｃｂｃ）前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、前記プログラムは更に、（ｃｃ）前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットであるかのいずれかであることを立証し、それにより前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とする検査装置が提供される。

【００３２】本発明の他の態様では、構文要素を含むＶＲＤに照らして構文要素を含むマークアップ言語文書の妥当性を検査する検査装置であって、（ａ）プロセッサと、（ｂ）（ｉ）前記文書と、（ｉｉ）前記ＶＲＤと、（ｉｉｉ）文書の妥当性を検査する手順を前記プロセッサに実行させるプログラムと、を格納するためのメモリを有し、（ｃ）前記プログラムは、（ｃａ）前記ＶＲＤを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、（ｃａａ）前記構造要素の構文属性を決定するコードと、（ｃａｂ）前記構造要素のハッシュ表現を決定するコードと、（ｃａｃ）前記ＶＲＤの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、前記プログラムは更に、（ｃｂ）マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、（ｃｂａ）前記文書構造要素の構文属性を決定するコードと、（ｃｂｂ）前記文書構造要素のハッシュ表現を決定するコードと、（ｃｂｃ）前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、前記プログラムは更に、（ｃｃ）前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現を、前記ＶＲＤの前記構造表現における対応構文属性及びハッシュ表現と比較し、それにより前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とする検査装置が提供される。

【００３３】本発明の他の態様では、ＶＲＤに照らしてマークアップ言語文書の妥当性検査を行う検査方法であって、前記ＶＲＤにおける第１タイプの最も深くネストされた構文要素についての第１拡張ハッシュ表現を決定するステップと、ＶＲＤリストに前記第１拡張ハッシュ表現を格納するステップと、マークアップ言語文書における第１タイプの最も深くネストされた構文要素について、第２拡張ハッシュ表現を決定するステップと、前記第２拡張ハッシュ表現が前記ＶＲＤリストに存在する場合に、前記マークアップ言語文書は妥当でないことを宣言するステップと、を含むことを特徴とする検査方法が提供される。

【００３４】本発明の他の態様では、ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、前記ＶＲＤの第１タイプの最も深くネストされた構文要素について第１拡張ハッシュ表現を決定する手段と、ＶＲＤリストの前記第１拡張ハッシュ表現を格納する手段と、前記マークアップ言語文書における第１タイプの最も深くネストされた構文要素について、第２拡張ハッシュ表現を決定する手段と、前記第２拡張ハッシュ表現が前記ＶＲＤリストに存在する場合に、前記マークアップ言語文書は妥当でないと宣言する手段と、を有することを特徴とする検査装置が提供される。

【００３５】本発明の他の態様では、ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続をコンピュータに実行させるコンピュータプログラムであって、前記ＶＲＤの第１タイプの最も深くネストされた構文要素について第１拡張ハッシュ表現を決定するコードと、ＶＲＤリストの前記第１拡張ハッシュ表現を格納するコードと、前記マークアップ言語文書における第１タイプの最も深くネストされた構文要素について、第２拡張ハッシュ表現を決定するコードと、前記第２拡張ハッシュ表現が前記ＶＲＤリストに存在する場合に、前記マークアップ言語文書は妥当でないと宣言するコードと、を含むことを特徴とするコンピュータプログラムが提供される。

【００３６】本発明の他の態様では、ＶＲＤに照らしてマークアップ言語文書の妥当性を検査する手続をコンピュータに実行させるコンピュータプログラムを格納したコンピュータに含まれるコンピュータプログラム製品であって、該プログラムは、前記ＶＲＤの第１タイプの最も深くネストされた構文要素について第１拡張ハッシュ表現を決定するコードと、ＶＲＤリストに前記第１拡張ハッシュ表現を格納するコードと、前記マークアップ言語文書における前記第１タイプの最も深くネストされた構文要素について第２拡張ハッシュ表現を決定するコードと、前記第２拡張ハッシュ表現が前記ＶＲＤリストに存在する場合に、前記マークアップ言語文書が妥当でないと宣言するコードと、を含むことを特徴とするコンピュータプログラム製品が提供される。

【００３７】

【ベストモードを含む詳細な説明】以下、添付の図面を参照して説明するが、ここでは、同じ符号を付したステップ及び／又は特徴は、逆の意図が表されない限り、同じ機能或は動作を有するものとする。

【００３８】本明細書に記載の発明の概念は、ＸＭＬタグについて、及び、可能ならばＸＭＬファイル中の他の要素について、”完全な”ハッシュを実行することによって、ＸＭＬパーサのメモリ必要量を減少させ、様々な性能値を向上させることができる、という考えに基づいている。ハッシュ関数とは、数学的であってもなくてもよいが、入力文字列を取得し、ハッシュ値と呼ばれる出力コード番号に変換する関数である。完全なハッシュ関数は、予めセットされた領域内のユニークな入力文字列に対して、ユニークなコード番号を作る関数である。入力文字列は、例えば、Ｗ３Ｃで許可された英数字や他の文字で構成され、ハッシュ処理の仕様に記されている特定の長さよりも短くなければならない。それに代えて、又は、それに加えて、入力文字列は、例えば、入力内容に基づいたコード番号の衝突が起こる可能性があるという点から、別の方法で制約を加えることができる。このアイディアによって、任意のＸＭＬタグを、数やコードとして扱うことができる。そして、メモリ内に数字の形で格納することができる。パーサは、通常、構造が解析されるときに、メモリ内のＸＭＬ構造の一部分を保存するので、ＸＭＬタグをユニークな数字に変換することによってメモリ必要量を減少させることができ、更に、文字列と文字列の比較を、同様の、しかしより高速の数値の比較に置き換えることができる。

【００３９】ここで記載する装置の原理は、非常に様々なマークアップ言語を用いた文書の解析にも、普通に応用することができる。説明を簡潔にするため、ここで開示される装置は、ＸＭＬ言語を参照して、説明することとする。しかし、これは発明の技術的範囲を制限しようとするものではない。例えば、ここに記載された機構はまた、UTF-16転送フォーマットに応用することができる（UTF-16の詳細は、国際規格ISO/IEC 10646-1を参照のこと）。

【００４０】図２Ａ及び図２Ｂは、従来技術のＳＡＸパーサ処理２３６を示したものである。これは、随意、整形性、及び／又は、妥当性をチェックするサブ処理をサポートする。

【００４１】図２Ａにおいて、マークアップ文書（本実施形態においては、ＸＭＬドキュメント）は、初期ステップ２００で開かれる。その後、決定ステップ２０２は、文書が未処理の（すなわち解析されていない）文字を含んでいるかどうかをテストし、決定ステップ２０２の条件に合ったなら、以下のステップ２０４で文字を読み込み、文字列に格納する。しかし、テストステップ２０２において、更に文字が検出されなければ、解析処理２３６はステップ２３４で終了する。

【００４２】ステップ２０４に続いて、テストステップ２０６において、チェックが行われ、完全な構文要素が組み立てられているかどうかを判断する。もし完全な構文要素が組み立てられているのであれば、パーサ処理２３６は”構文タイプ”識別ステップ２１０へ進む。反対に、完全な構文要素が組み立てられていなければ、パーサ処理２３６は文書中に使用できる文字が更に存在するかを判断する判断ブロック２０８へ向かう。さらに文字の使用が可能であれば、パーサ処理２３６は”はい”矢印に応じてステップ２０４へ戻る。一方、これ以上文字が使用できない場合は、処理２３６は”いいえ”矢印に応じて、構文要素タイプ識別ステップ２１０へ向かう。

【００４３】”タイプ識別”ステップ２１０は、組み立てられた構文要素に対して”タイプ”を識別し、その後、ステップ２１２で、要素文字列が文書構造のメモリ表現の中に置かれ、それによりこの時点で組み立てられていたように表現を増やす。その文章構造のメモリ表現は、一般的に、イベント駆動型パーサの場合、文書の部分構造表現であり、単純なリストとなり得る。

【００４４】ステップ２１２の後、処理２３６は整形性チェックを実行すべきかどうかを判断するテストステップ２４２へ向かう。整形性チェックは、文章が、インターネット上の http://www.w3.org/tr/2000/rec-XML-20001006.html で入手できる『拡張可能なマークアップ言語（ＸＭＬ）1.0（第２版）Ｗ３Ｃ勧告、2000年10月６日』の５ページに定義されているような、適切な”整形式制約”を満たすことをチェックする。整形性チェックは、文章が一般的な構造ルールに準拠しているかどうかをテストする、特に、文書中のタグが適切にネストになっているかどうかをテストする。そのようなチェックが実行されると、処理２３６は”ＹＥＳ”矢印に応じて、点線で示された境界線２４６上の”ａ”に向かう。参照文字”ａ”から”ｄ”に沿った点線２４６は、図２Ｂの対応する境界線に映されており、そこで処理２３６がさらに示されている。整形性チェックが実行されなかった場合、処理２３６は”いいえ”矢印に応じて、テストステップ２４２から”妥当性チェック”を実行すべきかどうかを判断するテストステップ２４４へ向かう。妥当性チェックは、前記Ｗ３Ｃ勧告５．１節に示される文書型定義（ＤＴＤ）のような、妥当性参照文書（簡単のためＶＲＤと称す）に定義される妥当性の制約に照らして、文書中の構文要素の比較を伴う。ＤＴＤとＸＭＬスキーマは妥当性チェックが実行されるＶＲＤの例であるが、ここで述べられている妥当性チェックは他のタイプのＶＲＤに対して実行される。この比較処理は、単なる整形性チェックに比し、より広範囲に渡って、要素の正しい構文配置を検証する。

【００４５】妥当性チェックが実行されると、処理２３６は”ＹＥＳ”矢印に応じて、点線で示された境界線２４６上の”ｂ”に向かう。他方、妥当性チェックが実行

されなかった場合、処理２３６は”ＮＯ”矢印に応じて、点線で示された境界線２４６上の”ｃ”に向かう。整形性チェックが選択された場合、処理２３６は境界線２４６上の”ａ”からオプションサブ処理２３８、特にその中の整形性チェックステップ２１４へ向かう。処理２３８のオプションとしての性質は、点線で囲まれた長方形で示される。

【００４６】妥当性チェックが選択された場合、処理２３６は境界線２４６上の”ｂ”からオプションサブ処理２４０、特にその中に見られる妥当性チェックステップ２２０へ向かう。処理２４０の選択的な性質は点線で囲まれた長方形で示される。妥当性チェックが選択されない場合、処理２３６は境界線２４６上の”ｃ”からアクション選択ステップ２２６へ向かう。整形性チェックが選択されて、整形性ステップ２１４の後、次の誤りチェックステップ２１６で誤りが検出された場合、矢印２１８で示されるように修正動作及び／または誤り指示が行われる。他方、誤りが検出されなければ、パーサ処理２３６はステップ２１６からサブ処理２４０へ向かい、そこでは妥当性チェックがステップ２２０で実行される。前述のように、解析処理２３６は、誤りチェックステップ２１６から、もしくは整形性チェックサブ処理２３８をバイパスして、妥当性チェックステップ２２０に向かい、処理２３６は境界線２４６上の“ｂ”から妥当性チェックステップ２２０に直接向かう。オプションの整形性サブ処理２３８は、適切な判断がテストステップ２４２，２４４で下された場合にバイパスすることができる（図２Ａ参照）。

【００４７】前述のように、妥当性チェックステップ２２０は、文書型定義（ＤＴＤ）に対して考慮されるマークアップ文書中の識別された構文要素の比較を含む。この比較処理は、サブ処理２３８に示されている単なる整形性チェックよりも広範囲に渡って、要素の正しい構文配置を検証する。

【００４８】妥当性チェックステップ２２０に続いて、誤りチェックステップ２２２で誤りが検出されると、修正動作が実行され、矢印２２４で示されるようにはい／いいえ誤り指示が実行される。一方、誤りが検出されない場合は、パーサ処理２３６はアクション選択処理２２６へ進み、そこでは考慮されている構文要素のタイプに基づいてアクションが選択される。図２Ａのステップ２４２及び２４４の決定において適切な判断が下されると、選択的サブ処理２３８及び２４０の両方をバイパスすることができる。両方の上記サブ処理がバイパスされると、前述のようにパース処理２３６は境界線２４６上の“ｃ”から直接アクション選択ステップ２２６へ向かう。

【００４９】構文要素がタグであった場合、矢印２２８で示されるように、タグ値や対応する文字列は、解析処理が実行されているアプリケーションに送られ、タグのメモリ表現が保存される。他方、要素タイプがタグのタイプでない場合、矢印２３０で示されるように、要素値文字列は関連アプリケーションに送られ、要素のメモリ表現は削除される。最終的に、パース処理２３６は、矢印２３２で示されるように、図２Ｂの点線で示された境界線上の“ｄ”へ向かい、図２Ａの対応する点線で示された境界線２４６上の“ｄ”から文字テストステップ２０２へ向かう。

【００５０】ＸＭＬ文書の冗長性により、多くのメモリ必要量が生じ、それに対応して、元の文字列形式中の文書構造を格納するために大量のメモリ必要量を要することになる。この文書構造はステップ２１２において参照される。さらに、可変長の英数字文字列間での文字列比較のパフォーマンスに関し、かなりの処理負荷が整形性チェックステップ２１４及び妥当性チェックステップ２２０の両方で生じる。

【００５１】次に示す２つのステップの両方に関連して、一般的に、文書の部分メモリ表現は格納されなければならず、文字列のチェックは実行されなければならない。つまり、（ｉ）階層分岐が閉じているかどうかのチェック、すなわち終了タグと開始タグのマッチング、また、分岐の重複がないかどうかのチェックについてのステップ２１４と、（ｉｉ）ＤＴＤに対する構造の一致性やタグ名のチェックに加えて、（ｉ）と同様の処理が必要となるステップ２２０と、である。パーサは、通常、ＸＭＬ構造が解析される場合、メモリ中にＸＭＬ構造の一部分を保存しなければならない。ＳＡＸパーサの場合でも、正しく動作するために、ＸＭＬ構造の局所的な部分がメモリ中に格納されなければならない。しかし、各ＸＭＬタグがハッシュ関数を用いてユニークな数字に変換される場合、ハッシュ演算で生じる数字は、関連する任意の長さのＸＭＬタグ文字列よりも小さいので、メモリ必要量は一般的に減少する。さらに、開始＆終了タグのマッチングに必要な文字列間の比較は、より高速な数値の比較で置き換えることができるため、処理負荷が減少する。

【００５２】代表的ハッシュアルゴリズムは、（ｉ）巡回冗長符号化アルゴリズム（ＣＲＣ）（通例、データ転送や格納において、署名分析又は誤り検出／訂正に用いられる）と、（ｉｉ）完全ロスレス符号化アルゴリズムと、（ｉｉｉ）ホフマン符号化アルゴリズムと、を含む。

【００５３】一般的に、適切なハッシュアルゴリズムは、静的な演算でなければならない。言い換えれば、要求されたデータセットに対し、同一の入力条件においては常に同じハッシュ結果を出力しなければならない。しかし、要求されたデータセットはその環境によって変化しうる。そして、そのようなデータセットは、通常、少なくとも一つの全マークアップ文書を含むことができ、また、関連ＤＴＤ或いはＸＭＬスキーマや、リンクされ

たマークアップ文書や、ＸＭＬ文書で参照されたＣＳＳ文書のような、異なった言語における関連の又はリンクされたマークアップ文書を含むこともできる。一方、タグ構文に遭遇した場合には、例えば、ＸＭＬ 文書に非リテラル‘<:’文字を見つけた場合には、常に、そのアルゴリズムをリセットすることによって、必要に応じて静的ハッシュアルゴリズムを使用することができる。また、ＸＭＬ ＤＴＤ文書に<:!ELEMENT文字列を見つけた場合に、又は、ＣＳＳ文書中であって有効なタグセレクタが許される全ての場合に、ハッシュアルゴリズムをリセットすることができる。

【００５４】入力されたマークアップ文書中のリファレンスは、適切なハッシュアルゴリズムに信号を送るか又はそれを選択するために用いられる。これは他のマークアップ文書や、ＤＴＤや、スタイルシートや、文字エンコーディングや、名前空間などを参照できるのとちょうど同じように実行されうる。例えば、特定のハッシュアルゴリズムは特定の名前空間と同じものとして扱うことができる。これにより、文書中の名前空間のリファレンスを介したハッシュアルゴリズムへの間接的なリファレンスが可能となる。ハッシュアルゴリズムの実行は、関連するパラメータ化と共に、全体的に又は部分的にマークアップ言語に含まれる。このような、ハッシュアルゴリズムを参照したり含めたりする方法は、最適化に有効である。つまり、特定のマークアップ文書に関して様々なハッシュ方法が最適化される場合である。それにより、宛先装置やシステム中の性能とメモリ使用量が改善される。一方、前述の参照方法がマッチングに有用な場合がある。それは１つ以上のマークアップ文書を伴うアプリケーションであり、そのアプリケーションで、誤りのチェック又は解析又は他の機能の完了が必要で、１つ以上の他の文書（例えば、ＤＴＤ）が既に同じアルゴリズムでハッシュ化されている場合である。

【００５５】上記アプローチでは、更なる改善が可能であり、例えば、ＤＴＤハッシュを任意的に含むことが挙げられる。これは、（より低速な）文字列の比較に代えて、数値の比較を可能とすることにより、ＤＴＤを格納するための読み出し専用メモリ（ＲＯＭ）の必要量を減少させ、ＸＭＬ文書の妥当性処理を高速化することができる。

【００５６】図３Ａ、３Ｂ及び３Ｃは、改良されたＳＡＸパーサ処理３４４のアレンジの１つを示したものである。図３Ａにおいて、ステップ３００からステップ３１０は、図２Ａに関して説明してきたステップ２００からステップ２１０に対応するものである。ステップ３１０の後、テストステップ３１２において、組み立てられた構文要素の性質がタグであるか、別の要素タイプであるか確かめるためにテストされる。要素がタグの場合は、パース処理３４４は矢印３１６に従って、ハッシュステップ３１８へ向かう。ハッシュステップ３１８は、それぞれ図５及び６のプロセッサ４１４及び５０５を用いて、構文要素の単一数値表現を決定する。これによって、要素の表現を効率良くメモリ上に実現され、また、英数字領域よりも、単純で高速な数値での比較演算ができる。構文要素を示す要素文字列とそのハッシュ値の両方が、処理３４４の時点で保持されるが、ステップ３１８で、各々のメモリ４１８及び５０６（図５及び６を参照）を用いて文書構造のメモリ表現へ挿入されるのは、文字列の値ではなくハッシュ値である。ここまではこれのほうがよい。

【００５７】図３Ａ、図３Ｂ及び図３Ｃで示されるようなパース処理３４４を、より正確に理解するため、まず、図２で示したパース処理２３６との関係で、ＸＭＬ部の例［１］を解析する場合について説明する。この場合、サブ処理２１２において、ＸＭＬ部［１］から、以下のような解析されたマークアップタグの階層表現が生成される。

【００５８】

```
205  Shakespeare
215      div
220        mult
221        /mult
225      banquo
235            quote
240            quote
245      /banquo
250        Hamlet
251            quote
252            /quote
253        /Hamlet
255  /Shakespeare
```

［２］

【００５９】逆に、図３Ａに示されるような、パース処理３４４において、タグ要素と非タグ要素とに対して異なる処理が行われた結果、ステップ３１８で同等な階層表現が生成される。その同等階層表現は［３］に示される。［３］の階層表現は、解析されハッシュにされたマークアップタグから成る。これを例示するために、タグ

名の領域は、以下の表１に示されるように制約され、ハッシュマッピング（機能的にはハッシュ“関数”の適用と同等である）は次の表で示される。

【００６０】
【表１】

タグ	ハッシュコード番号
Shakespeare	133
Div	326
Mult	371
Banquo	787
Quote	629
Hamlet	411

【００６１】上記ハッシュマッピングに基づき、［１］で示されたＸＭＬ文書は、以下のような階層表現となる。

```
205  133
215       326
222         371
223         /371
225       787
235                 629
240                 /629
245       /787
254         411
255                   629
256                   /629
257         /411
255  /133
```

【００６２】

［３］

【００６３】図３Ａに戻ると、パース処理３４４は、ステップ３１４から点線で示された境界線３５６上の“ａ”へ向かう。参照文字“ａ”及び“ｂ”に沿った点線３５６は、図３Ｂの対応する境界線に映されており、それに関連して処理３４４が更に示されている。

【００６４】図３Ｂに移り、処理３４４は、点線で示された境界線３５６上の“ａ”から、整形性チェックが実行されるかどうかを判断するテストステップ３５０へ続く。そのチェックが実行されると、処理３４４は「ＹＥＳ」矢印に応じて、境界線３５８上の“ｃ”へ向かう。参照文字“ｃ”及び“ｆ”に沿って点線で示された境界線３５８は、図３Ｃの対応する境界線に映されており、それに関連して処理３４４が更に示されている。整形性チェックが実行されなかった場合、処理３４４は、「ＮＯ」矢印に応じて、妥当性チェックが実行されるかどうかを判断するテストステップ３５２へ向かう。妥当性チェックが実行されると、処理３４４は「ＹＥＳ」矢印に応じて、境界線５８上の“ｅ”へ向かう。一方、妥当性チェックが実行されなかった場合は、処理３４４は境界線３５８上の“ｄ”へ向かう。

【００６５】図３Ｃに向かうと、整形性チェックが実行された場合、処理３４４は、点線で示された境界線３５８上の“ｃ”から、整形性チェックステップ３２０へ向かう。一方、整形性チェックが選択されなかった場合、処理３４４は、点線で示された境界線３５８上の“ｅ”から、妥当性チェックステップ３２６へ進む。整形性チェックと妥当性チェックのどちらも選択されなかった場合、処理３４４は、点線で示された境界線３５８上の“ｄ”から、アクション選択ステップ３３４へ進む。

【００６６】整形性チェックステップ３２０は、対応するプロセッサ４１４及び５０５を用いて、整形性チェックを実行し、選択的処理３４６の一部分を形成している。処理３４６の選択的性質は点線を用いて示される。同様に、妥当性ステップ３２６はサブ処理３４８の一部分を形成し、その選択的性質は点線で示されている。

【００６７】［３］に表される階層表現により、文字列の比較がより高速でより効率的な数値の比較に置き換えられることは明らかである。それに従って関連する計算の負荷が軽減される。さらに、［３］で示される階層表現は、［１］で示される階層表現よりもメモリ効率に優れており、それに対応して［３］で示される表現は、前述したようなメモリ制約のあるアプリケーションにより適している。

【００６８】図３Ｃに戻り、整形性チェックが選択され、ステップ３２０で整形性チェックが実行された後、パース処理３４４は誤りチェックステップ３２２へ向かう。ここでは、誤りが検出されると、矢印３２４で示されるように、修正動作が実行され、及び/又は、誤りが指示される。整形性チェックは、文書中のタグが適切にネストになっているかどうかを一般的に考慮する。従って、例えば、［２］を参照すると、“Hamlet”と“/Hamlet”のタグペアは“Shakespeare”と“/Shakespeare”

のタグペア内に適切にネストになっている。なぜなら、“Hamlet”タグペアは、完全に“Shakespeare”タグペア内にネストになっており、そのタグペア同士は、例えば、互いに重なり合っていないからである。

【００６９】一方、誤りが検出されなかった場合、パース処理３４４はオプションの処理３４８へ向かう。ここにおいて、ＤＴＤ及びＸＭＬスキーマに関して、対応するプロセッサ４１４及び５０５を用いた、妥当性チェックステップ３２６が実行される。前述のように、妥当性チェックは整形性チェックよりも詳細にチェックを行うものである。従って、例えば、整形性チェックは、“Hamlet”のタグのペアが“Shakespeare”のタグペアの間に正しくネストになっているとみなすのに対し、妥当性チェックは、これとは異なり、“Hamlet”のタグペアが“Shakespeare”のタグペアの間に完全にネストになっているという正しいネストのチェックだけでなく、“Hamlet”のタグペアがこの方法で、正当なネストになっているかどうかのチェックも行う。例えば、実際、“Shakespeare”のタグペアが“Hamlet”のタグのペアの間にネストにならなければならず、この逆のネストではだめな場合もある。従って、妥当性チェック処理は、この場合、ネストが適切であるかどうか、すなわち完全に実行されるかどうかに加え、“Hamlet”タグペアが“Shakespeare”のタグペアの間にネストになり得るかどうか、というタグの階層表現もチェックする。

【００７０】妥当性ステップ３２６を実行するため、ＤＴＤ／ＸＭＬスキーマのメモリ表現を、ハッシュステップ３１８で生成されたマークアップ文書のハッシュの性質に一致するように、ＤＴＤ又はＸＭＬスキーマタグは、ハッシュステップ３２８で最初にハッシュにしておく。妥当性チェックステップ３２６は、ステップ３１８で生成された文書の構造表現と、ステップ３２８で生成されたＤＴＤ／ＸＭＬスキーマの構造表現とを比較する。そして、マークアップ文書の構文要素の正しい構文配置を検証する。なお、図２のステップ２２０で用いられるように、この比較で必要な文字列の比較は、ステップ３２８及び３１８のハッシュ演算の結果、ここで、図３Ｃにおいて、より高速でより効率的な数値の比較に置き換えられる。

【００７１】妥当性チェックの後、処理３４４は誤りチェックステップ３３０へ向かう。ここでは、矢印３３２で示されるように、修正動作及び/または誤り指示が実行される。誤りがなかった場合、パース処理３４４は、アクションステップ３３４へ進む。ここにおいて、構文要素のタイプがタグの場合、パース処理が実行されているアプリケーションに、対応するタグの文字列が送られ、タグの文字列自体は、図５及び６にある４１８又は５１２のメモリから消去される。しかし、関連するハッシュで表されたタグのメモリ表現は保持される。従って、現在解析されているタグの文字列のコピーを除けば、タグの文字型のメモリ表現は保持されない。タグのメモリ表現は、これによりハッシュの形式だけとなる。要素の構文タイプが非タグタイプの場合、又は非タグ名タイプの場合、矢印３３８で示されるように、要素の値又は文字列の表現は、関連のアプリケーションへ送られ、関連のメモリ表現は消去される。パース処理３４４は今、矢印３４０で示されるように、点線で示された境界線３５８上の“ｆ”へと折り返し、その後図３Ｂの点線で示された境界線３５８上の対応する“ｆ”へ向かい、その後点線で示された境界線３５８上の“ｂ”へ向かい、その後図３Ａの点線で示された境界線３５６上の対応する“ｂ”へ向かい、最後に文字テストステップ３０２へと向かう。これ以上文字が検出されなければ、パース処理３４４はステップ３４２で終了する。

【００７２】ＸＭＬ文書部［１］は、ハッシュ形式のタグを持つようになり、以下のような形式で示される。

【００７３】

```
505 <:133>:
110 <:!—This is a comment—>:
515 <:326 class="preface"Name1="value1"name2="value2">:
520 <:371 list=&:lt;>: <:/371>:
525 <:787>:
130 Say
535 <:629>:
540 goodnight<:/629>:,
545 Hamlet. <:/787>:
550 <:411>:<:629>:Goodnight,Hamlet.<:/629>:<:/411>:
555 <:/133>:
```

［４］

【００７４】終了タグの表現は（一般的には、開始タグに<:section>:という書式を用いるのに対し、<:/section>:という書式を用いる）、様々な方法で定義することができ、それによってＸＭＬ標準仕様との互換性が高くなったり、低くなったりする。開始タグと終了タグは、現在の記述において、“同等のタイプ”だとみなされる点に注意しなければならない。さらに、開始タグと終了タグは一まとめにして機能する、すなわち文書内容の部分を区切ることは、２つのタグに関連性があることを意味する。さらに、開始タグと終了タグの前記書式は、終了タ

グが開始タグを修飾していることを意味し、識別文字、すなわち“/”といったものは、対応する終了タグを作り出すために開始タグに組み込まれている点に注意しなければならない。ＸＭＬ標準仕様との互換性は、場合によっては重要となることがある。好適な実施形態において、現在のタグ文字列の‘/’文字は、開始タグと終了タグの名前がハッシュ関数から同じ数値が返されるために、次のタグの名前をハッシュにする前に、一般的には取り除かれる。ＸＭＬタグの例は、インターネット上のhttp://www.w3.org/tr/2000/rec-XML-20001006.htmlで入手できる『拡張可能なマークアップ言語（ＸＭＬ）1.0（第２版）Ｗ３Ｃ勧告、2000年10月６日』の3.1節に見られるように、<:/Name Attribute>:によって示される。現在の記述では、“タグ”は、文脈によって、考慮している特定のタグの一部分又は全体のいずれかを参照する。一方、メモリ中では‘/’文字（終了タグを識別する）に相当する表現を保持することが必要となる場合もありうる。これは、以下のさまざまな方法で行われる。例えば、（ｉ）終了タグであることを示すために、終了タグのハッシュ数字の近傍で、‘/’文字又はそれに相当する文字をメモリに「再挿入」すること、又は（ｉｉ）ハッシュタグの始まりか終わりの状態を示すブール値を用いること、又は、（ｉｉｉ）完全にマッチングした場合、開始タグと終了タグの単純な和がゼロとなるように、終了タグのハッシュ値を負にすること、等が挙げられる。（ｉｉｉ）の場合、開始タグのハッシュは、本実施形態において、必要な終了タグのハッシュを作り出すために、単純な否定演算子によって変更される。（ｉｉｉ）は、符号ビットがハッシュアルゴリズムに影響を受けないという保証が必要である。事実上、（ｉｉｉ）はブールフラグオプション（ｉｉ）とほとんど同じである。

【００７５】さらに、構造化されたハッシュ番号が作られ、そこではネストになったタグ用のハッシュ番号は、第１のタグがネストになっているところではより高レベルのＸＭＬタグであることを明示できる。従って、例えば、タグ123がタグ987の内部にネストになっているところでは、ネストになったタグ123のように示す代わりに、987.123のように示すことができる。この構造化されたハッシュ又は“拡張された”ハッシュにより、またがった構造の演算が減少する。すなわち、タグのペアの終了点を検索している間にメモリに保持しなければならないＸＭＬ文書の量を減らすことによって解析の性能をさらに向上することができる。

【００７６】拡張された表現は、ハッシュに基づく必要はなく、文字列や“数え上げ法”に基づくことができるという点にも注意しなければならない。数え上げ法は、タグの名前と数字の間で配置が定義され、目録表や索引が作られる処理である。数え上げ法の簡単な形式は、全てのタグの名前を単にリストにし、列記されたタグに番号を付けたものである。従って、例えば、"Shakespeare.banquo.quote"の形式の連結文字列は、３つの連結タグの文字型拡張表現を表す。

【００７７】ＸＭＬ部［３］に対する構造化された同等のハッシュにされたマークアップの例は、負の終了タグのハッシュを用いた、以下の［５］で示される。

【００７８】

```
133
133.326
133.371
133.-371
133.787
133.787.629          ->013307870629
133.787.-629         ->-013307870629
133.787                                    [5]
133.411
133.411.629
133.411.-629
133.-411
-133
```

【００７９】［５］において、ネストになったタグの構造は、［３］で示される形式から、一連の連結ハッシュタグに変換される。ここでは、各々の後続の低レベルの（すなわち、より深くネストされた）ハッシュタグの階層レベルは、前の上段の階層レベルに関連付けられる。これはハッシュで表されたＤＴＤから、同様に解析された構造を用いて、簡単な数の比較を実行することができる。実際、［５］の各々の行は、［５］の行４及び行５で示されるように、出くわした一連のハッシュタグを連結して組み合わさった１つの数字で表される。この１つの数字は、元の入力タグの識別と関係を非常に圧縮された形で表し、それによって同様にハッシュで表されたＤＴＤと非常に効率のよい比較ができる。数字のタグセットは、文書構造を表すために、非常に圧縮された形で用いることができることがわかる。妥当性チェックは、単にハッシュで表された開始タグのセットを用いるだけで

実行でき、各々のこのセットは、文書構造の各々の枝の最も深い、全体の、構造を表す点に注意が必要である。例えば、［５］の構造を最小限に表したものは、以下の

01330326
01330371
013307870629
013304110629

【００８１】また、ＤＴＤ又はＸＭＬスキーマ構造も同じ方法によって表すことができる。

【００６６】解析とハッシュが行われた入力文書からのタグセットと、解析とハッシュが行われたＤＴＤの間の１つ又は複数の数値の比較は、ＸＭＬパーサの妥当性に通常必要とされる一連の文字列及び構造の比較を置き換える。ＤＴＤ又はＸＭＬスキーマによって定義されたどんな代替構造も、入力ＸＭＬ文書から生じたハッシュタグセット数字と後で比較するために用いる単一ハッシュタグセット数字にコード化できることがわかる。

【００８２】図４は、例えば、ＤＴＤ又はＸＭＬスキーマのようなＶＲＤに対して、マークアップ文書を検証するための処理６００を示したものである。処理６００は、検証されるマークアップ文書が開かれるステップ６０２から始まる。その後、ステップ６０４で、現在の拡張タグは、図５又は図６の各々のプロセッサ４１４又は５０５によってリセットされる。図４の説明において、“タグ”、“拡張タグ”、“テンポラリータグ”等は、各々のタグのハッシュ表現を指す。以下のステップ６０６では、テンポラリータグのルートは、各々のプロセッサ４１４又は５０５の１つによってリセットされ、その後、マークアップ文書中の次のタグが、ステップ６０８で識別される。その後、テストステップ６１０は、ステップ６０８で識別されたタグが開始タグであるかどうか判断する。テストステップ６１０のイベントでは、処理６００は「ＹＥＳ」矢印に応じてステップ６１２へ向かう。ステップ６１２では、ステップ６０８で識別されたタグを、各々のプロセッサ４１４又は５０５のうち１つを用いた拡張タグに加える。次に、処理６００はステップ６１２からステップ６０８へ戻る。

【００８３】テストステップ６１０が、次のタグは開始タグではないと判断した場合、処理６００は「ＮＯ」矢印に応じて、拡張タグが“０”かどうかを判断し文書のルートレベルを表すテストステップ６１４へ向かう。“０”の値が検出された場合、処理６００は「ＹＥＳ」矢印に応じて、文書の終わりに到達したかどうかを判断するテストステップ６２４へ向かう。もしそうでない場合には、処理６００は「ＮＯ」矢印に応じてステップ６０６へ向かう。なお、ステップ６１４で“０”の値が検出されるのは、開始タグと終了タグの数が合わない構造等のように、整形性のない文書構造だからかもしれない。

【００８４】テストステップ６１４が、拡張タグの値が“０”でないことを判断した場合、処理は「ＮＯ」矢印に応じて、拡張タグがテンポラリータグのルート値と等しいかどうかを判断するテストステップ６１６へ向かう。そうでない場合には、処理６００は「ＮＯ」矢印に応じて、各々のメモリ４１８及び５０６中に文書リスト中の拡張タグを格納するステップ６１８へ進む。一方、テストステップ６１６が、拡張タグがテンポラリータグのルートに等しいことを判断した場合、処理６００は各々のプロセッサ４１４と５０５を用いて、「ＹＥＳ」矢印に応じて拡張タグから最も低い（すなわち最も深いネストである）タグを取り除くステップ６２０へ向かう。その後、処理６００は、ステップ６２２で拡張タグをテンポラリータグのルートにコピーし、処理はステップ６０８へ戻る。

【００８５】テストステップ６２４に戻る前に、これまで記述されていたように、処理６００は正当性がチェックされるマークアップ文書へ向かう点に注意しなければならない。しかし、明確には記述されていないが、妥当性参照文書（ＶＲＤ）に適用される同一の処理も存在し、この処理によって処理６００により作り出される文書リストをテストできるＶＲＤリストが作り出される。処理６００及びＶＲＤに向かうそれに相当する処理は、異なる時間に通常発生する。処理６００は、検証される全ての文書に対して発生し、各々の特定の文書を検証するために拡張したハッシュ表現のリストを作り出す。ＶＲＤリストがステップ６２６の前に完成される場合、ＶＲＤリストは処理６００と実質的には同時に作り出すことができる。一方、ＶＲＤ処理はオフラインで実行することができ、結果のリストはステップ６２６の前に処理６００に与えられる。

【００８６】ステップ６２６に戻ると、前述のようにＶＲＤリストが入手できるので、ステップ６２６は文書リスト中の全ての入力がＶＲＤリストに見つかるかどうかを判断する。もしそうである場合には、処理６００は「ＹＥＳ」矢印に応じて、文書は妥当であると宣言するステップ６２８に向かう。一方、文書リストがＶＲＤリストには見られない入力を持つ場合、処理６００は「ＮＯ」矢印に応じて、文書が無効であると宣言するステップ６３０に向かう。

【００８７】ステップ６２６でさらに詳しく述べられているように、前記記述では、文書リストの全ての入力とＶＲＤリストの全ての入力を比較する。もう１つの処理は、ステップ６１６の後に、ステップ６２６と同様のス

［６］のように示される。

【００８０】

［６］

テップでの完全なＶＲＤリストに対して、各々の拡張タグをテストすることである。この処理のイベントでは、文書拡張タグがＶＲＤリストに見つからなかった場合、これ以上不要なテストを行わずに、処理６００がステップ６３０に直接進むことができる。しかし、拡張タグが見つかった場合、処理６００はステップ６２０等に向かうことができる。この処理によって誤りを早く認識し、ただちに妥当性処理を終了することができる。ＶＲＤリ

```
01
    02
        03
        -03
        04
        -04
    -02
    05
    -05
-01
```

【００９０】機能面から言えば、処理６００はマークアップ文書の階層構造の最も深い枝の部分に降りる。すなわち［７］の第１行にある“01”から［７］の第３行にある“03”へ向う。そして、その特定の枝の最も深い部分を表す拡張ハッシュ表現、すなわち“010203”を格納する。次に、この処理は、新たに探索すべき枝を示す別の開始タグ（この例の場合、［７］の５行目にある“04”）を見つけるまで、終了タグを捨て、枝を横切る。処理が新しい枝に降りる場合、その処理は、階層のより高いレベルにある拡張ハッシュ表現を保存する。これらのハッシュ表現は、処理がこれらのレベルに戻るまで保存される。文書構造中の誤り（無効の文書、又は整形性されていない文書）は、これらのＶＲＤにマッチングしない拡張ハッシュ表現を通常返す。ステップ６２０は、前の開始タグに対して検索された終了タグの整形性チェックを任意的に含んでもよく、その結果、文書に整形性があれば整形性マッチを提供する。前出の開始タグは、拡張ハッシュ表現の中で最低のタグである点に注意すべきである。例えば、ＤＴＤ/ＸＭＬスキーマは、ステップ６１０で、ステップ６２０の拡張ハッシュ表現にある最低の開始タグにマッチングしない終了タグを返し、それによって整形性チェックにひっかかることになるかもしれない。

【００９１】ステップ６２６でのテストは、一般的に、その文書のＤＴＤ用にリストになったハッシュ表現に対して、マークアップ文書構造の拡張ハッシュ表現をマッチングさせようとする。文書構造のハッシュ表現は、一般的にＤＴＤリストにある最も深い構造表現のサブセットである。従って、妥当なＸＭＬ文書は、対応するＶＲＤ又はＤＴＤ中に定義されている、あらゆる正当な構造的ネストのサブセットを含んでよい。従って、ステップ６２６における一般的なテストでは、ＤＴＤのより深いハッシュ表現と、文書のより浅いハッシュ構造表現との比較を含む。従って、例えば、ＸＭＬ文書からの拡張ハッシュ表現“0123”は、対応するＤＴＤからのハッシュ表現“01230456”と比較した場合、「妥当性がある」と評価される。

【００９２】図４に示される妥当性処理６００は、高速だが不完全な妥当性チェックを用いて、最も深くネストになった部分等の、文書構造中のより複雑な部分をチェックするように、最適化されうる。従って、任意の性能特性を有する特定の妥当なパーサを実行するために、速度と妥当性の感度の最適な組み合わせを選択できる。

【００９３】また、妥当性方法６００を変更して「標準的な」整形性チェックの少なくとも一部分を実行することもできる。従って、例えばステップ６２０において、拡張タグ表現の中で、最も低い階層にあるタグ表現のハッシュに対して、終了タグのハッシュ表現をチェックすることができる。前記表現が、同じオリジナルのタグであると適切に識別されなかった場合、その文書は整形性がなく、回復や誤り処理が実行される。

【００９４】前記方法は、構造チェックとは別にまたは一緒に、構造内で定義された属性のハッシュ表現を含むように拡張できる。

【００９５】この妥当性及び整形性チェックの方法を、別々の処理における入力文書や、文書構造及び内容を解析する処理に対して、適用できることは明らかである。従って、例えば、方法６００は、中央演算処理装置（ＣＰＵ）サイクル及びメモリサイズが特に大きな制約を受けていない環境においても、妥当性及び整形性チェックを、効率良く、高速に実行できるようにするために、最適化することができる。そのようなシステムで別々のチェックを実行する利点として、高度に最適化されたチェックを用いて、「妥当性のない」文書を直ちに廃棄するストが短ければさらに効率がアップする。完全な妥当性チェックが、前記方法で実行される場合、ステップ６２８は考慮している文書が妥当であることを示す。

【００８８】前述の妥当性の方法をさらに説明するために、以下の構造部（ここでは開始タグが“01”から“05”であり、対応する終了タグはそれぞれ“-01”から“-05”である）を考える。

【００８９】

［７］

ことができるという事実が挙げられる。これは、少なくとも妥当でない文書の一部分にかける時間と処理を大幅に節約することができる。それによって、例えば、(i)文章が妥当でないことを見つけるためだけに、文章を解析して完全なＤＯＭツリー表現にし、妥当性チェックを実行すること、又は(ii)妥当でない文書の第２の部分を検出する前に、文書の第１の（妥当な）部分の処理を更に開始すること（文書の第１部分の処理がそれによって無駄になっている）を防ぐことができる。もう１つの利点は、高速な妥当性チェックを用いて文書に妥当性がないとわかった後、次のジョブの処理を直ちに始められる点である。

【００９６】「不完全な」ハッシュ処理、すなわち、各々の英数字の入力文字列に対して単一の数字を作りだす保証がないハッシュ処理にも、適した場合があり、特に、ＸＭＬタグ文字列の最大長が制限されている、又はある確率で少なくとも制限されている場合に適している。さらに、ＸＭＬタグ文字列セットが限られた数の文字順列に制限されるか、又は限られた数の文字順列に制限される可能性がある場合、適切に動作するために不完全なハッシュ処理が設計され、選択される。

【００９７】文書構造の数的表現のための通信規格、又は、代替の公用及び私的フォーマットは、ハッシュアルゴリズムの使用に基づいて、定義され、記述される。この技術によって、冗長性のために大量のデータ伝送を通常伴う、ＸＭＬデータの伝送にメリットとなる圧縮形式及び人間が読み易いASCII形式が可能となる。人間の読み易さを保持するか、しないかに対して、様々な選択肢が存在する。例えば、（完全な）ハッシュを、他の圧縮形式と組合わせることが挙げられる。これは、ＸＭＬファイル中の様々な要素タイプにそれぞれ適用される例えば、ＸＭＬ文字列タグを、完全なハッシュアルゴリズムから得られる、ユニークで、人間が読みやすい数字に、置き換えることができる。また、ハッシュではない構文要素及び他の要素は、処理又は装置間で、伝送用の非損失圧縮技術によって圧縮することができ、それによって、伝送データ量が減少する。

【００９８】前段落で述べたように、必要なところに、逆ハッシュアルゴリズム又は可逆ハッシュアルゴリズムを参照したり、含んだりすることができる。これは、例えば、ハッシュされる前の、伝送された、マークアップ文書から、表示したり、ラベリングしたりする目的で、１つ以上のマークアップタグを人間が読み易い文字列にデコード、又は解読するために、そのようなアルゴリズムが必要とされる場合に用いられる。ここでは、別に、解析や誤りチェックのためにそのようなことを行う必要はない。可逆ハッシュアルゴリズム、又は逆ハッシュアルゴリズムのもう１つ利用法は、マークアップタグ又は他のデータを解読することによって、伝送されたマークアップ文書について制限された機能や特徴が使用できるようになることである。可逆アルゴリズム、又は逆アルゴリズムは、マークアップ文書の送信部と受信部のマッチングに使用することができる。可逆ハッシュアルゴリズム、又は逆ハッシュアルゴリズムは、受信部に既に含まれているので伝送はされないが、マークアップ文書で参照される場合もある。可逆ハッシュアルゴリズム、又は可逆ハッシュアルゴリズムには、(i) 完全非損失アルゴリズム、及び(ii)ハフマン符号化、が例として挙げられる。

【００９９】上記機構は、以下の条件の１つ以上が当てはまる場合には、あらゆるマークアップ言語に対して、非常に有利な立場で適用することができる。その条件とは、すなわち、(i) マークアップ言語はタグ名の定義（例えば、ＸＭＬ、ＤＴＤ、ＣＳＳ、ＸＳＬ等）を許す、(ii)タグ名は大きな文字コードテーブル（例えば、UTF-16）を使用し、及び/又は、タグ名の長さは一般的にそのハッシュ表現よりも短くない、(iii)マークアップ文書を用いる又は受け取る使用アプリケーションは、マークアップ文書、ＸＭＬスキーマ又はＤＴＤの中に最低でも１つのネストの階層レベルを持った、複雑な構造表現を通常必要とする、(iv)入力マークアップ文書は、何らかチェック形式、通常は整形性又は妥当性チェックが必要である、(v) マークアップパーサ、及び/又は、アプリケーションは、メモリ容量（例えば、組み込み式又は低価格ＣＰＵシステム）又はメモリ管理（例えば、仮想メモリがないシステム、又はダイナミックメモリが割り当てられていないシステム）に厳しい制限がある、(vi)複雑で、高度にネストなる可能性があるマークアップ文書上で素早く動作する必要があるマークアップパーサ及び/又は、アプリケーションである。

【０１００】本明細書で示されたマークアップ言語文書の解析方法は、マークアップ言語文書の解析用関数、又はサブ関数を実行する、１つ以上の集積回路等の専用ハードウェアで実行できる。そのような専用ハードウェアは、画像プロセッサ、ディジタル信号プロセッサ、又は１つ以上のマイクロプロセッサ及び関連メモリを有していてもよい。マークアップ言語文書の解析方法は、一方、専用組込みコンピュータシステム４００を用いて、実行することができる。そのようなシステムは図５で示される。このシステムにおいて、図３Ａ、図３Ｂ、図３Ｃ、及び図４の処理は、組込みコンピュータシステム４００中で実行されるアプリケーションプログラム等のソフトウェアとして実行されてもよい。コンピュータシステム４００は、典型的には、プリンタ（図には示されていない）等のエンドシステムに組み込まれ、プリンタ中のプリンタエンジン４０２を動作させる。特に、組込みコンピュータによって実行されるソフトウェアの命令によって、マークアップ言語の解析方法のステップが行われる。そのソフトウェアは、読み出し専用メモリ（ＲＯＭ）４１８、ランダムアクセスムモリ（ＲＡＭ）４１

８、又は他のタイプのメモリ（示されていない）を含んだコンピュータ可読媒体に収納されてもよい。そのソフトウェアは、製造時に組込みコンピュータにロードされるか、或は、その場でソフトウェアのアップグレードが行われる。

【０１０１】組込みコンピュータシステム４００は、コンピュータモジュール４１０、パラメタ設定用のスイッチモジュール４２２等の入力装置、ジョブ状態を示す液晶表示器（ＬＣＤ）等の出力装置、プリンタエンジン４０２で構成される。組込みコンピュータ４００は、通常、物理的にプリンタ（示されていない）に組み込まれる。コンピュータネットワーク４０４に接続された他のコンピュータ（示されていない）で起こった印刷ジョブは、入出力（Ｉ／Ｏ）インターフェース４０８への接続４０４によって、組み込みコンピュータ４００に送られる。

【０１０２】組込みコンピュータモジュール４１０は、プロセッサユニット４１４及びメモリユニット４１８、更に、例えば、半導体ランダムアクセスメモリ（ＲＡＭ）、読み込み専用メモリ（ＲＯＭ）、スイッチモジュールとＬＣＤインターフェース４１６を含む入出力（Ｉ／Ｏ）インターフェース、プリンタエンジン４０２とネットワーク４０６用の入出力インターフェース４０８で構成される。組込みコンピュータ４１０の構成要素４０８及び構成要素４１４～４１８は、通常相互接続されたバス４１２を介して交信しており、これは関連技術で知られる組込みコンピュータシステム４１０の従来動作モードのようになる。一般的に、この機構に関するプログラムは、メモリ４１８に格納され、プログラム動作時にプロセッサ４１４がプログラムを読み込んで制御する。

【０１０３】また、マークアップ言語文書の解析方法は、図６で示されるような、従来型汎用コンピュータシステム５００を用いて実行することができる。このシステムにおいて、図３Ａ、図３Ｂ、図３Ｃ、及び図４の処理は、コンピュータシステム５００中で動作するアプリケーションプログラム等のソフトウェアとして実行されてもよい。このアプリケーションは、例えば、ハッシュが、ネットワークを介したコンピュータ間の通信規格として用いられる場合に役立つ。図６は、現在考慮している通信コンピュータの一例を示したものである。

【０１０４】特に、マークアップ言語文書の解析方法のステップは、コンピュータによって実行されるソフトウェアの命令によって行われる。そのソフトウェアは２つの分離部品に分けられてもよい。すなわち、１つは解析方法を実行する部分で、もう１つは後者とユーザの間のユーザーインターフェースを管理する部分である。例えば、そのソフトウェアは、以下に示す記憶装置を有するコンピュータ可読媒体に格納されてもよい。そのソフトウェアはコンピュータ可読媒体からコンピュータにロードされ、次にコンピュータにより実行される。そのようなソフトウェアを有するコンピュータ可読媒体、又は前記媒体に記録されたコンピュータプログラムは、コンピュータプログラム製品である。このコンピュータプログラム製品のコンピュータでの使用は、本発明の実施形態に応じて、マークアップ言語文書を解析に有利な装置で実行されるのが望ましい。

【０１０５】コンピュータシステム５００は、コンピュータモジュール５０１、キーボード５０２及びマウス５０３等の入力装置、プリンタ５１５及びディスプレイ装置５１４を含む出力装置で構成される。変調・復調（モデム）送受信装置５１６は、例えば、電話線５２１又は他の機能媒体を介して接続可能な通信ネットワーク５２０へ双方向で通信するために、コンピュータモジュール５０１で用いられる。モデム５１６は、インターネットや他のネットワークシステム、例えば、ローカルエリアネットワーク（ＬＡＮ）又はワイドエリアネットワーク（ＷＡＮ）、コンピュータ５００と通信している他のパーソナルコンピュータ（ＰＣ）５２２等へアクセスできるようにするために用いられる。

【０１０６】コンピュータモジュール５０１は、典型的には、少なくとも１つのプロセッサユニット５０５、メモリユニット５０６を含み、例えば、半導体ランダムアクセスメモリ（ＲＡＭ）、読み込み専用メモリ（ＲＯＭ）、ビデオインターフェース５０７を含んだ入力・出力（Ｉ／Ｏ）インターフェース、キーボード５０２及びマウス５０３用の入力・出力インターフェース５１３、及びオプションとしてのジョイスティック（図で示されていない）、モデム５１６のためのインターフェース５０８で構成される。

【０１０７】記憶装置５０９が設けられており、通常、ハードディスクドライブ５１０及びフロッピー（登録商標）ディスクドライブ５１１を含む。また、磁気テープドライブ（不図示）を用いてもよい。ＣＤ－ＲＯＭドライブ５１２は、非揮発性のデータ源として通常備えられる。コンピュータモジュール５０１の構成要素５０５～５１３は、相互接続されたバス５０４を介して通常交信し、これは関連技術で知られるコンピュータシステム５００の従来動作モードのようになる。本実施形態において実行できるコンピュータには、IBM PC、IBM PC互換機、及びSun Sparcstations又はそこから発展した同様のシステムが例に挙げられる。

【０１０８】典型的には、本実施形態におけるアプリケーションプログラムは、ハードディスクドライブ５１０上に格納され、プログラム動作時にプロセッサ５０５がプログラムを読み込んで制御する。ネットワーク５２０から取り出されたプログラム及びあらゆるデータの中間記憶装置は、場合によってはハードディスクドライブ５１０と連動して、半導体メモリ５０６を用いて実現してもよい。場合によっては、アプリケーションプログラムは、ＣＤ－ＲＯＭ又はフロッピーディスクにエンコード

してユーザに供給されてもよく、対応するドライブ５１２又は５１１を介して読み込まれるか、代わりに、モデム装置５１６を介してネットワーク５２０上でＰＣ５２２からユーザによって読まれてもよい。

【０１０９】さらに、磁気テープ、ＲＯＭ、集積回路、磁気光ディスク、ラジオ又はコンピュータモジュール５０１及び別の装置間の赤外線トランスミッションチャンネル、ＰＣＭＣＩＡカード等のコンピュータ可読カード、電子メールの送信及びウエブサイトやそれと同様のものを有するインターネット及びイントラネットを含む他のコンピュータ可読媒体から、コンピュータシステム５００にソフトウェアをロードすることもできる。上記は、コンピュータ可読媒体に関連した例に過ぎない。発明の範疇や技術的思想から外れない限り、他のコンピュータ可読媒体は、実行してもよい。

【０１１０】（産業上の利用分野）以上から、本発明の実施形態は、コンピュータ及びデータ処理産業に適用できることは明らかである。上記は、本発明の実施形態のいくつかを示したに過ぎず、発明の範囲や意図から外れない限り、修正及び/又は変更を行うことができる。上記実施形態は、例に過ぎず、限定的なものではない。

【図面の簡単な説明】

【図１Ａ】本発明の好適な実施形態に係るＸＭＬパーサシステムのブロック図である。

【図１Ｂ】本発明の好適な実施形態に係るＸＭＬパーサシステムのブロック図である。

【図２Ａ】選択的な整形性チェック及び/又は妥当性チェックを含む、従来技術におけるＳＡＸパーサの方法ステップを示すフローチャートである。

【図２Ｂ】選択的な整形性チェック及び/又は妥当性チェックを含む、従来技術におけるＳＡＸパーサの方法ステップを示すフローチャートである。

【図３Ａ】図２Ａ及び図２ＢのＳＡＸパーサを改良した機構を示す図である。

【図３Ｂ】図２Ａ及び図２ＢのＳＡＸパーサを改良した機構を示す図である。

【図３Ｃ】図２Ａ及び図２ＢのＳＡＸパーサを改良した機構を示す図である。

【図４】ＤＴＤ又はＸＭＬスキーマ等の参照文書に対する、文書の妥当性の処理について示す図である。

【図５】改良されたＳＡＸパーサの機構を実行できる専用組込みコンピュータのブロック図である。

【図６】改良されたＳＡＸパーサの機構を実行できる汎用コンピュータを示す図である。

【図１Ａ】

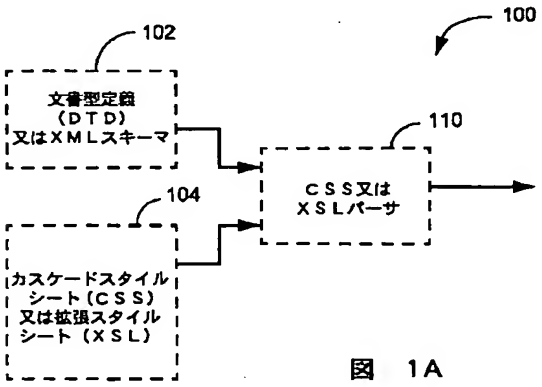


図　1A

【図１Ｂ】

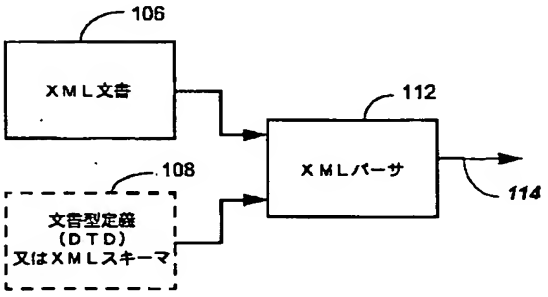


図　1B

【図２Ａ】

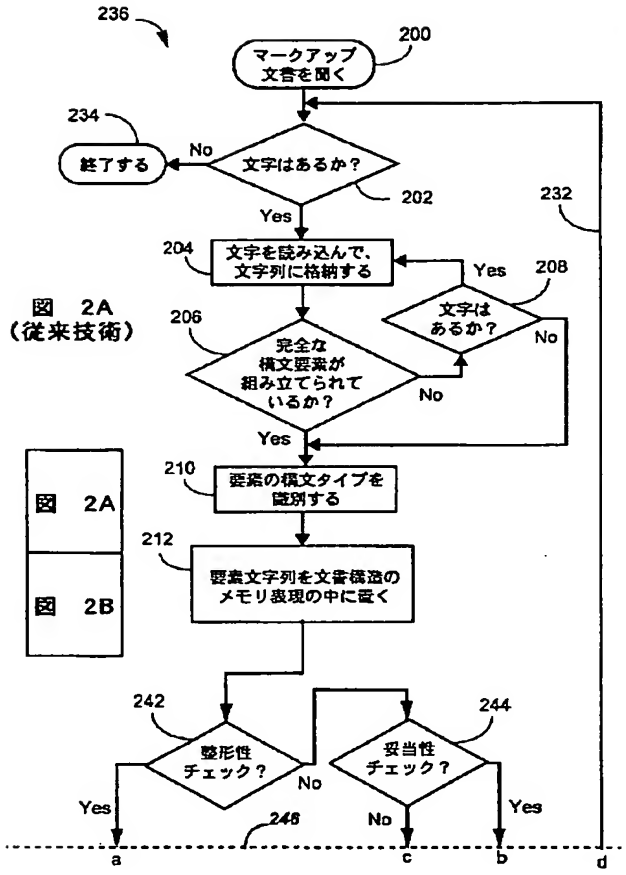


図 2A
（従来技術）

【図２Ｂ】

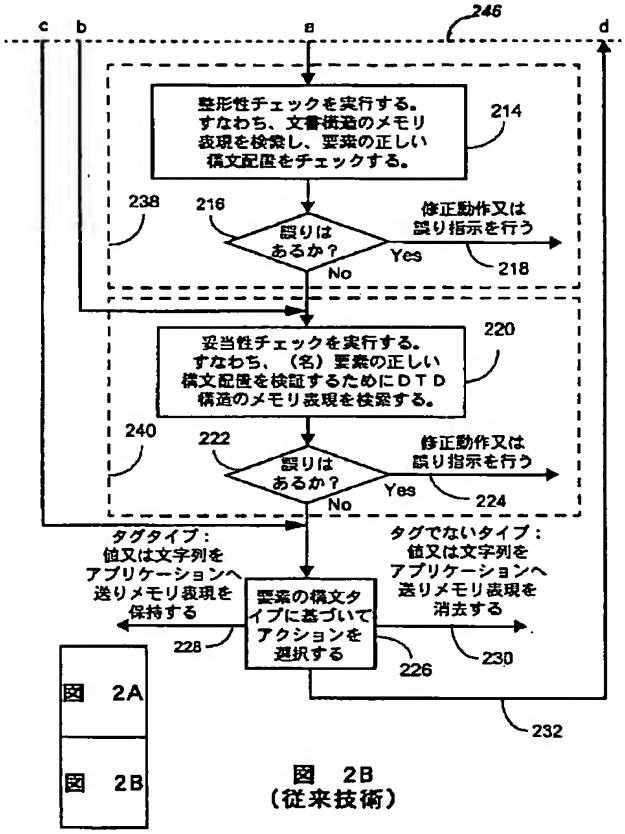


図 2B
（従来技術）

【図３Ａ】

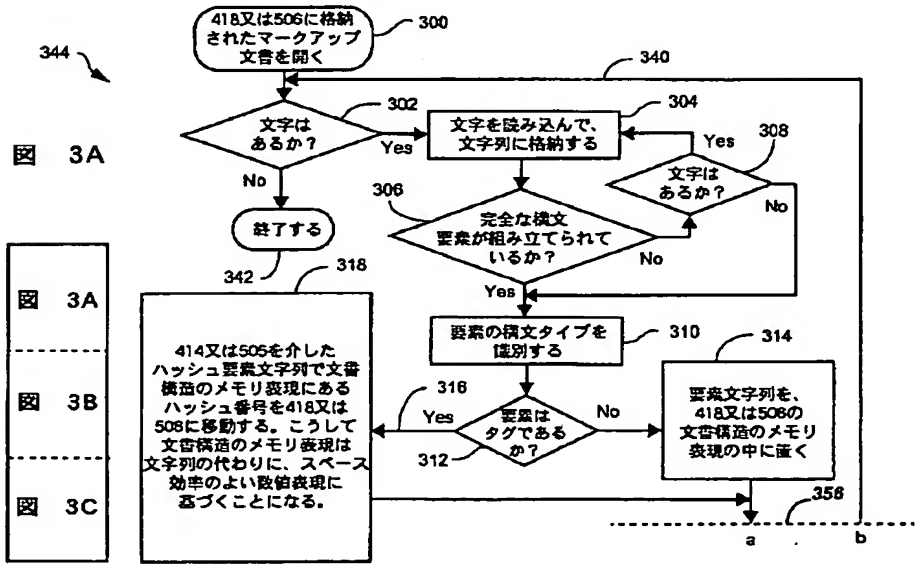


図 3A

【図３Ｂ】

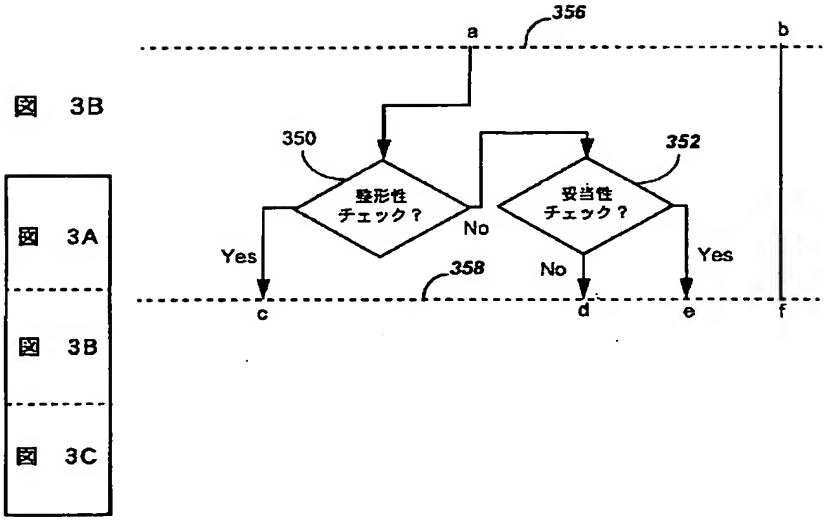


図 3B
図 3A
図 3B
図 3C
356
a
b
350
352
整形性チェック？
妥当性チェック？
No
Yes
No
Yes
358
c
d
e
f

【図３Ｃ】

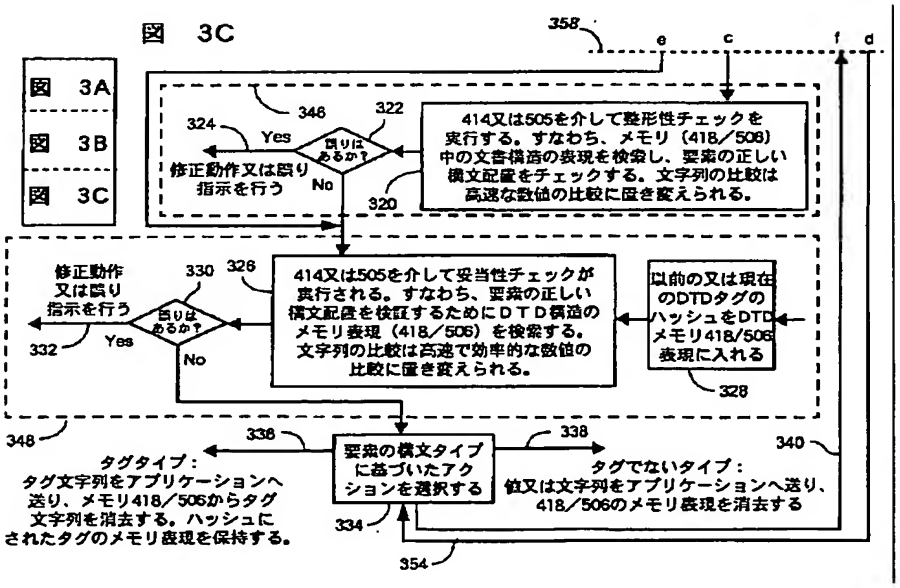


図 3C
図 3A
図 3B
図 3C
358
e
c
f
d
346
322
324
Yes
誤りはあるか？
No
修正動作又は誤り指示を行う
414又は505を介して整形性チェックを実行する。すなわち、メモリ（418／508）中の文書構造の表現を検索し、要素の正しい構文配置をチェックする。文字列の比較は高速な数値の比較に置き変えられる。
320
修正動作又は誤り指示を行う
330
326
誤りはあるか？
332
Yes
No
414又は505を介して妥当性チェックが実行される。すなわち、要素の正しい構文配置を検証するためにＤＴＤ構造のメモリ表現（418／506）を検索する。文字列の比較は高速で効率的な数値の比較に置き変えられる。
以前の又は現在のDTDタグのハッシュをDTDメモリ418/506表現に入れる
328
348
336
338
340
タグタイプ：タグ文字列をアプリケーションへ送り、メモリ418／506からタグ文字列を消去する。ハッシュにされたタグのメモリ表現を保持する。
要素の構文タイプに基づいたアクションを選択する
タグでないタイプ：値又は文字列をアプリケーションへ送り、418／506のメモリ表現を消去する
334
354

【図４】

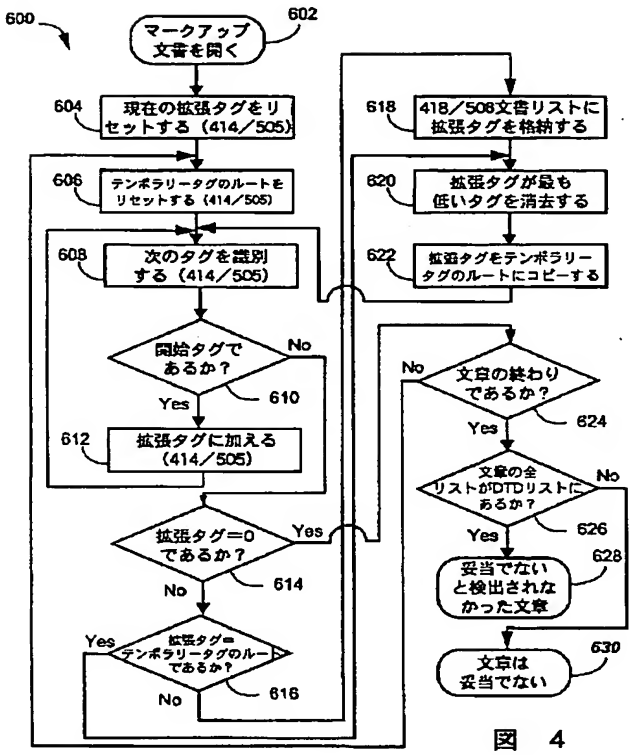

600
マークアップ文書を開く
602
現在の拡張タグをリセットする（414／505）
604
テンポラリータグのルートをリセットする（414／505）
606
次のタグを識別する（414／505）
608
開始タグであるか？
No
Yes
610
拡張タグに加える（414／505）
612
拡張タグ＝0であるか？
Yes
No
614
拡張タグ＝テンポラリータグのルートであるか？
Yes
No
616
418／508文書リストに拡張タグを格納する
618
拡張タグが最も低いタグを消去する
620
拡張タグをテンポラリータグのルートにコピーする
622
文章の終わりであるか？
No
Yes
624
文章の全リストがDTDリストにあるか？
No
Yes
626
妥当でないと検出されなかった文章
628
文章は妥当でない
630

図　4

【図５】

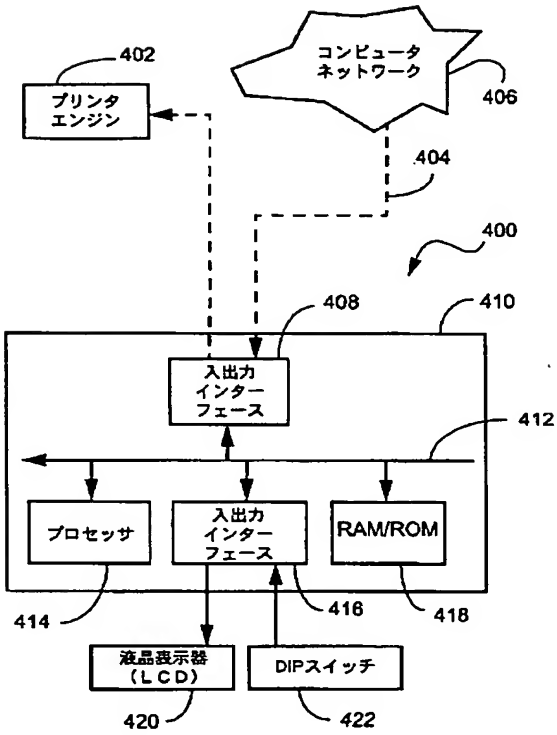

402
プリンタエンジン
コンピュータネットワーク
406
404
400
408
410
入出力インターフェース
412
プロセッサ
入出力インターフェース
RAM/ROM
414
416
418
液晶表示器（ＬＣＤ）
DIPスイッチ
420
422

図　5

【図６】

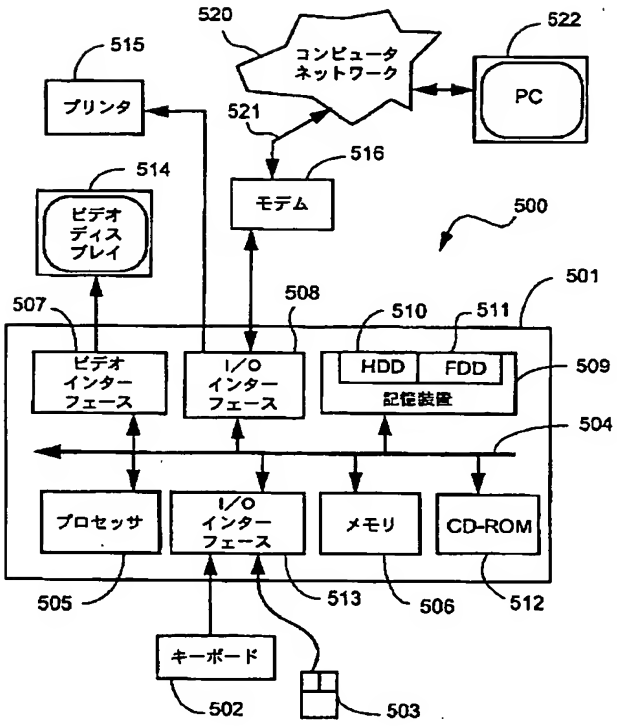

520
コンピュータ
ネットワーク
522
PC
515
プリンタ
521
516
モデム
514
ビデオ
ディス
プレイ
500
501
507
508
510
511
ビデオ
インター
フェース
I/O
インター
フェース
HDD
FDD
記憶装置
509
504
プロセッサ
I/O
インター
フェース
メモリ
CD-ROM
505
513
506
512
キーボード
502
503

図 6

【外国語明細書】

1. Title of the Invention

HASH COMPACT XML PARSER

2. Claims:

1. A method of parsing a markup language document comprising syntactic elements, said method comprising, for one of said syntactic elements, the steps of:

identifying a type of the element;

processing the element by determining a hash representation thereof if said type is a first type; and

augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

2. A method according to claim 1, wherein said parsing is event-based parsing.

3. A method according to claim 1, wherein said hash representation is determined using one of:

a hash algorithm;

a first reference to said hash algorithm dependent upon an associated Universal Reference Indicator;

a second reference to said hash algorithm dependent upon an associated namespace; and

a third reference to said hash algorithm dependent upon an associated Extended Markup Language declaration;

4. A method according to claim 2, wherein said first type is one of:

one of a structural element and a part thereof;

a definition of said structural element;

a declaration of said structural element; and

a match for said structural element.

5. A method according to claim 4, wherein said structural element is a tag.

6. A method according to claim 2, wherein the hash representation is a unique code for said one syntactic element, said element having less than a first number of characters.

7. A method according to claim 2, wherein the hash representation is not a unique code for said one syntactic element, said element being constrained, to a probability level, in terms of at least one of (i) a number of characters in the element and (ii) a permissible number of permutations of characters in the element.

8. A method according to claim 6, wherein said code comprises numeric characters.

9. A method according to claim 2, wherein said processing step comprises a sub-step of:

determining an extended hash representation of both (i) said one syntactic element being a first instance of said first type, and (ii) another syntactic element being a second instance of said first type, within which said first instance, said second instance is nested.

10. A method according to claim 1 comprising, for another one of said syntactic elements, further steps of:

identifying a type of the other element; and if the type of the other element is equivalent to said first type:

(i) processing the other element to thereby determine a second hash representation thereof; and

(ii) augmenting said at least partial structural representation of the document using the second hash representation, wherein:

said processing and said second processing ensure that if a first relationship exists between the one element and the other element, then a second relationship which is representative of the first relationship, exists between the hash representation of the one element and the hash representation of the other element.

11. A method according to claim 10, wherein:

the one element is a start tag;

the other element is an end tag;

the hash representation of the one element is a corresponding hashed start tag, and;

the second hash representation of the other element is a corresponding hashed end tag.

12. A method according to claim 11, wherein:

the end tag is a first modification of the start tag; and

the hashed end tag is a second modification of the hashed start tag, said second modification being representative of the first modification.

13. A method according to claim 12, wherein:

the end tag is the same as the start tag apart from having a distinguishing character incorporated therein; and

the hashed end tag is at least one of:

the same as the hashed start tag;

the same as the hashed start tag apart from having a distinguishing character incorporated therein; and

the hashed start tag having been processed by an operator.

14. A method according to claim 12, wherein:

the one and the other element comprise respectively a start tag and an end tag, being a first pair of tags;

corresponding hashed start and end tags for said first pair of tags are incorporated into the partial structural representation of said document;

the document further includes a second pair of tags comprising a respective start tag and end tag, said second pair of tags being nested within said first pair of tags in said document, said method comprising further steps of:

processing said second pair of tags to form corresponding second hashed start and end tags;

augmenting said at least partial structural representation of the document using said corresponding second hashed start and end tags so that said second hashed start and end tags indicate a nesting in relation to said hashed start and end tags for the first pair of tags which is equivalent to the nesting of said second pair of tags within said first pair of tags.

15. A method according to claim 14 comprising, prior to said augmenting step, a further step of:

concatenating the first hashed start tag with the second hashed start tag, and concatenating the first hashed end tag with the second hashed end tag, to thereby form respective extended hashed start and end tags for said second pair, wherein:

said augmenting step is performed using said respective extended hashed start and end tags for said second pair, and:

said extended hashed start and end tags indicate a nesting in relation to said hashed start and end tags for the first pair of tags which is equivalent to the nesting of said second pair of tags within said first pair of tags.

16. A method according to claim 1, wherein the augmenting step is succeeded by a well-formedness checking step against a syntactic rule, said well-formedness checking step comprising checking said at least partial structural representation of the markup language document against the syntactic rule by numerically

comparing corresponding hashed representations of elements in said at least partial structural representation of the markup language document

17. A method according to claim 16, wherein said numerically comparing step is succeeded by a further step of:

string-comparing, in accordance with said syntactic rule, corresponding non-hashed representations of elements not of said first type.

18. A method according to claim 1, wherein said first type is one of:

one of a structural element and a part thereof;

a definition of said structural element;

a declaration of said structural element; and

a match of said structural element.

19. A method according to claim 18, wherein said structural element is a tag.

20. A method according to claim 14, comprising a further step of:

checking the well-formedness of said at least partial structural representation of the document against a syntactic rule.

21. A method according to claim 20, wherein the syntactic rule relates to proper nesting of tags and said checking step comprises sub-steps of:

performing a numerical comparison across hashed tags in said at least partial structural representation of the document to thereby identify said first hashed start and end tags and said second hashed start and end tags; and

verifying that the second hashed start and end tags indicate a proper nesting in relation to said first hashed start and end tags.

22. A method according to claim 21, wherein the numerical comparison is followed by a further step of:

performing a string comparison, in accordance with said syntactic rule, across non-hashed parts of respective tags in said at least partial structural representation of the document .

23. A method according to claim 15, comprising a further step of:

checking the well-formedness of said at least partial structural representation of the document against a syntactic rule.

24. A method according to claim 23, wherein the syntactic rule relates to proper nesting of tags and said checking step comprises sub-steps of:

performing a numerical comparison across hashed tags in said at least partial structural representation of the document to thereby identify said first hashed start and end tags and said extended hashed start and end tags; and

verifying that the extended hashed start and end tags indicate a proper nesting in relation to said first hashed start and end tags.

25. A method according to claim 24, wherein the numerical comparison is followed by a further step of:

performing a string comparison across non-hashed parts of respective tags in said at least partial structural representation of the document.

26. A method according to claim 16, wherein the well-formedness checking step is one of (a) succeeded by, (b) included in, and (c) replaced by a validation step against a validation reference document VRD, said validation step comprising sub-steps of:

(a) processing the VRD, said processing comprising, for a syntactic element in the VRD, sub-sub-steps of:

(i) identifying a type of said syntactic element of the VRD; and

(ii) processing the syntactic element by determining a hash representation thereof if said type is said first type; and

(b) checking said at least partial structural representation of the markup language document against the processed VRD, said checking comprising a sub-sub-step of numerically comparing corresponding hashed representations of elements.

27. A method of validating a markup language document against a VRD, said method comprising steps of:

(a) processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said processing comprising steps of:

(i) determining a hierarchy position of said document tag;

(ii) determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(b) processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said processing comprising steps of:

(i) determining a hierarchy position of said tag;

(ii) determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) storing said extended hashed representation of said tag in a list; and

(c) validating said markup language document if said extended hashed representation of said document tag is one of found in said list and is a valid subset of a member of said list.

28. A method of validating a markup language document against a VRD, said method comprising steps of:

(a) processing said VRD, for each structural element identified therein, said processing comprising steps of:

(i) determining syntactic attributes of said structural element;

(ii) determining a hashed representation of said structural element; and

(iii) storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) processing the markup language document, for each document structural element identified therein, said processing comprising steps of:

(i) determining syntactic attributes of said document structural element;

(ii) determining a hashed representation of said document structural element; and

(iii) storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

29. A method according to claim 26, wherein said numerically comparing step is succeeded by a further step of string-comparing corresponding non-hashed representations of elements not of said first type.

30. A method according to claim 26, wherein said first type is one of:

one of a structural element and a part thereof;

a definition of said structural element;

a declaration of said structural element; and

a match of said structural element.

31. A method according to claim 30, wherein said structural element is a tag.

32. A method of encoding a markup language document comprising syntactic elements, said method comprising, for one of said syntactic elements, steps of:

identifying a type of the syntactic element; and

processing the syntactic element by one of:

(i) determining a hash representation thereof if said type is a first type;

(ii) determining a compressed representation thereof if said type is not a first type; and

(iii) retaining the syntactic element.

33. A method of decoding a markup language document comprising encoded syntactic elements, said method comprising, for one of said encoded syntactic elements, steps of:

identifying a type of the encoded syntactic element;

processing the encoded syntactic element by at least one of:

(i) determining an inverse hash representation thereof if said type is a first type; and

(ii) determining a decompressed representation thereof if said type is not a first type; and

(iii) retaining the encoded syntactic element.

34. An apparatus for parsing a markup language document comprising syntactic elements, said apparatus comprising:

identifying means for identifying a type of the element;

processing means for processing the element by determining a hash representation thereof if said type is a first type; and

augmenting means for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

35. An apparatus for validating a markup language document against a VRD, said apparatus comprising:

(a) means for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said means comprising:

(i) means for determining a hierarchy position of said document tag;

(ii) means for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) means for storing said extended hashed representation of said document tag if said document tag is more deeply nested than an extended hashed representation of a previous document tag;

(b) means for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said means comprising:

(i) means for determining a hierarchy position of said tag;

(ii) means for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) means for storing said extended hashed representation of said tag in a list; and

(c) means for establishing whether said extended hashed representation of said document tag is one of to be found in said list, and is a valid subset of a member of said list, thereby validating said markup language document.

36. An apparatus for validating a markup language document against a VRD, said apparatus comprising:

(a) means for processing said VRD, for each structural element identified therein, said means comprising:

(i) means for determining syntactic attributes of said structural element;

(ii) means for determining a hashed representation of said structural element; and

(iii) means for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) means for processing the markup language document, for each document structural element identified therein, said means comprising:

(i) means for determining syntactic attributes of said document structural element;

(ii) means for determining a hashed representation of said document structural element; and

(iii) means for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) means for comparing syntactic attributes and hashed representations of said each document structural element in the structural representation of the document to corresponding syntactic attributes and hashed representations in said structural representation of said VRD to thereby establish validity of the markup language document.

37. An apparatus for encoding a markup language document comprising syntactic elements, to form an at least partial structural representation of the document, said apparatus comprising:

means for identifying a type of the syntactic element; and

means for processing the syntactic element by one of:

(i) determining a hash representation thereof if said type is a first type;

(ii) determining a compressed representation thereof if said type is not a first type; and

(iii) retaining the syntactic element.

38. An apparatus for decoding a markup language document comprising encoded syntactic elements, said apparatus comprising:

means for identifying a type of the encoded syntactic element;

means for processing the encoded syntactic element by at least one of:

(i) determining an inverse hash representation thereof if said type is a first type;

(ii) determining a decompressed representation thereof if said type is not a first type; and

(iii) retaining the encoded syntactic element.

39. A computer program which is configured to make a computer execute a procedure to parse a markup language document comprising syntactic elements, said program comprising:

code for identifying a type of an element;

code for processing the element by determining a hash representation thereof if said type is a first type; and

code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

40. A computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said document tag;

(ii) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) code for storing said extended hashed representation of said document tag if said tag is more deeply nested than a previous document tag;

(b) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said tag;

(ii) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) code for storing said extended hashed representation of said tag in a list; and

(c) code for validating said markup language document if said extended hashed representation of said document tag is one of found in said list, and is a valid subset of a member of said list.

41. A computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing said VRD, for each structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said structural element;

(ii) code for determining a hashed representation of said structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said document structural element;

(ii) code for determining a hashed representation of said document structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) code for validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

42. A computer program which is configured to make a computer execute a procedure to encode a markup language document comprising syntactic elements, said program comprising:

code for identifying a type of the syntactic element; and

code for processing the syntactic element by one of:

(i) determining a hash representation thereof if said type is a first type;

(ii) determining a compressed representation thereof if said type is not a first type; and

(iii) retaining the syntactic element.

43. A computer program which is configured to make a computer execute a procedure to decode a markup language document comprising encoded syntactic elements, said program comprising:

code for identifying a type of the encoded syntactic element;

code for processing the encoded syntactic element by at least one of:

(i) determining an inverse hash representation thereof if said type is a first type; and

(ii) determining a decompressed representation thereof if said type is not a first type; and

(iii) retaining the encoded syntactic element.

44. A computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to parse a markup language document, said program comprising:

code for identifying a type of the element;

code for processing the element by determining a hash representation thereof if said type is a first type; and

code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

45. A computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said document tag;

(ii) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) code for storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(b) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said tag;

(ii) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) code for storing said extended hashed representation of said tag in a list; and

(c) code for validating said markup language document if said extended hashed representation of said document tag is one of found in said list and is a valid subset of a member of said list.

46. A computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing said VRD, for each structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said structural element;

(ii) code for determining a hashed representation of said structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said document structural element;

(ii) code for determining a hashed representation of said document structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) code for validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

47. An at least partial structural representation a markup language document comprising syntactic elements, said at least partial representation having been produced by a method comprising, for one of said syntactic elements, the steps of:

identifying a type of the element;

processing the element by determining a hash representation thereof if said type is a first type; and

augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

48. An apparatus for parsing a markup language document comprising syntactic elements, said apparatus comprising:

a processor;

a memory for storing (i) the document, and (ii) a program which is configured to make the processor execute a procedure to parse the document;

said program comprising:

(i) code for identifying a type of an element;

(ii) code for processing the element by determining a hash representation thereof if said type is a first type; and

(iii) code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

49. An apparatus for validating a markup language document comprising syntactic elements against a VRD comprising syntactic elements, said apparatus comprising:

(a) a processor;

(b) a memory for storing (i) the document, (ii) said VRD, and (iii) a program which is configured to make the processor execute a procedure to validate the document;

(c) said program comprising:

(ca) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(caa) code for determining a hierarchy position of said document tag;

(cab) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(cac) code for storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(cb) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said means comprising:

(cba) code for determining a hierarchy position of said tag;

(cbb) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(cbc) code for storing said extended hashed representation of said tag in a list; and

(cc) code for establishing whether said extended hashed representation of said document tag is one of to be found in said list, and is a valid subset of a member of said list, thereby validating said markup language document.

50. An apparatus for validating a markup language document containing syntactic elements against a VRD containing syntactic elements, said apparatus comprising:

(a) a processor;

(b) a memory for storing (i) the document, (ii) said VRD, and (iii) a program which is configured to make the processor execute a procedure to validate the document;

(c) said program comprising:

(ca) code for processing said VRD, for each structural element identified therein, said code comprising:

(caa) code for determining syntactic attributes of said structural element;

(cab) code for determining a hashed representation of said structural element; and

(cac) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(cb) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(caa) code for determining syntactic attributes of said document structural element;

(cab) code for determining a hashed representation of said document structural element; and

(cac) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(cc) code for comparing syntactic attributes and hashed representations of said each document structural element in the structural representation of the document to corresponding syntactic attributes and hashed representations in said structural representation of said VRD to thereby establish validity of the markup language document.

51. A method of validating a markup language document against a VRD, said method comprising steps of:

determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

storing said first extended hashed representation(s) in a VRD list;

determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

52. A method according to claim 51, wherein said syntactic element of said first type is one of:

one of a structural element and a part thereof;

a definition of said structural element;

a declaration of said structural element; and

a match for said structural element.

53. A method according to claim 52, wherein said structural element is a tag.

54. A method according to claim 51, wherein:

said most deeply nested syntactic elements(s) in the VRD are syntactic elements(s) which are most deeply nested within one of the global structure of the VRD and a local sub-structure of the VRD; and

said most deeply nested syntactic element in the markup language document is a syntactic element which is most deeply nested within one of the global structure of the markup language document and a local sub-structure of the markup language document.

55. An apparatus for validating a markup language document against a VRD, said apparatus comprising:

means for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

means for storing said first extended hashed representation(s) in a VRD list;

means for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

means for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

56. A computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

code for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

code for storing said first extended hashed representation(s) in a VRD list;

code for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

code for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

57. A computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

code for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

code for storing said first extended hashed representation(s) in a VRD list;

code for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

code for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

58. A method according to claim 7, wherein said code comprises numeric characters.

3. Detailed description of the Invention

Copyright Notice

Technical Field of the Invention

The present invention relates generally to processing of multimedia documents, and, in particular, to documents produced in mark-up language. The present invention relates to a method and apparatus for parsing documents in mark-up language. The invention also relates to a computer program and a computer program product including a computer readable medium having recorded thereon said computer program, which is configured to make a computer execute a procedure for parsing a document composed in a mark-up language.

Background Art

Parsing is a process of extracting information from a document. The process usually involves at least a minimum check of document syntax, and can in general yield either a tree structure description of the document, or a logical chain of events. The structural representation based on the logical chain of events is typically produced by an ordered parsing of the document from beginning to end.

Tree-based parsers compile, for example, an XML document into an internal tree structure, providing a hierarchical model which applications are able to navigate. The Document Object Model (DOM) working group at the World-Wide Web consortium is presently developing a standard tree-based Application Programming Interface (API) for Extended Markup Language (XML) documents. Event-based parsers, on the other hand, report parsing events such as the start and end of elements directly to the application for which the parsing is being performed. This reporting is performed typically using callbacks, and does not require an internal tree structure. The application requiring the parsing implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based parsers are useful for a wide range of applications, but typically place a strain on system resources, particularly if the document being parsed is large. Furthermore, applications sometimes need to build their own particular tree structures, and it is inefficient to build a tree representation, only to map it to a different representation. Event-based parsers provide a simpler, lower-level access to an XML document, facilitating parsing of documents larger than available system memory. The "Simple API for XML" (referred to as the SAX parser) is an event-driven interface for parsing XML documents. SAX parsers are discussed in more detail in relation to Figs. 2(a), 2(b), 3(a), 3(b) and 3(c).

Figs. 1(a) and 1(b) shows block representations of parser systems. The following XML document fragment 106 is considered:

```
105  <Shakespeare>
110  <!--This is a comment-->
115  <div class="preface" Name1="value1" name2="value2">
120  <mult list=&lt;> </mult>
125  <banquo>
130  Say .                                                    [1]
135  <quote>
140   goodnight  </quote>,
145  Hamlet.</banquo>
150  <Hamlet><quote>Goodnight, Hamlet. </quote></Hamlet>
155  </Shakespeare>
```

In Fig. 1(b), the XML document 106 is input into a parser 112 which, in the present instance, is an event based parser. Optionally, as indicated by a dashed box 108, a Document-Type-Definition (DTD) or an XML Schema is also input into the parser 112. The parser 112 outputs, as depicted by an arrow 114, a partial structural representation of the document 106 which can be a simple list. In Fig. 1(a), a Cascading Style Sheet (CSS) or an Extendable Style Sheet (XSL) 104 is input into a CSS or XSL parser 110. A DTD 102 can also be input into this parser 110. Both the XML parser 112 and the CSS/XSL parser 110 are event driven parsers in the present illustration.

One of the benefits of mark-up languages such as XML is the facility to make documents smarter, more portable and more powerful, by enabling the use of tags to define various parts of the documents. This capability derives from the descriptive nature

of XML. XML documents can be customised on a per-subject basis, and accordingly, customised tags can be used to make the documents comprehensible, in terms of the structure, to a human reader. This very attribute, however, often leads to XML documents being verbose and large, and this poses a problem in some instances. For example, where XML documents must be parsed in a hardware-constrained piece of equipment, such as a printer, the typically memory intensive nature of conventional parsing is in conflict with the limited memory which can be accommodated in such equipment. Furthermore, the human readability of XML documents is typically of minimal benefit when the documents are processed by hardware constrained pieces of equipment. Furthermore, tag-string matching operations, which are required to a significant degree in XML document parsing, pose a sometimes unacceptable burden of processing requirements, translating into an unacceptable number of processor cycles. These problems apply to both parser instances shown in Figs. 1(a) and 1(b).

Disclosure of the Invention

It is an object of the present invention to substantially overcome, or at least ameliorate, one or more disadvantages of existing arrangements.

According to a first aspect of the invention, there is provided a method of parsing a markup language document comprising syntactic elements, said method comprising, for one of said syntactic elements, the steps of:

identifying a type of the element;

processing the element by determining a hash representation thereof if said type is a first type; and

augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided a method of validating a markup language document against a VRD, said method comprising steps of:

(a) processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said processing comprising steps of:

(i) determining a hierarchy position of said document tag;

(ii) determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(b) processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said processing comprising steps of:

(i) determining a hierarchy position of said tag;

(ii) determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) storing said extended hashed representation of said tag in a list; and

(c) validating said markup language document if said extended hashed representation of said document tag is one of found in said list and is a valid subset of a member of said list.

According to another aspect of the invention there is provided a method of validating a markup language document against a VRD, said method comprising steps of:

(a) processing said VRD, for each structural element identified therein, said processing comprising steps of:

(i) determining syntactic attributes of said structural element;

(ii) determining a hashed representation of said structural element; and

(iii) storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) processing the markup language document, for each document structural element identified therein, said processing comprising steps of:

(i) determining syntactic attributes of said document structural element;

(ii) determining a hashed representation of said document structural element; and

(iii) storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

According to another aspect of the invention there is provided a method of encoding a markup language document comprising syntactic elements, said method comprising, for one of said syntactic elements, steps of:

identifying a type of the syntactic element; and

processing the syntactic element by one of:

(i) determining a hash representation thereof if said type is a first type;

(ii) determining a compressed representation thereof if said type is not a first type; and

(iii) retaining the syntactic element.

According to another aspect of the invention there is provided a method of decoding a markup language document comprising encoded syntactic elements, said method comprising, for one of said encoded syntactic elements, steps of:

identifying a type of the encoded syntactic element;

processing the encoded syntactic element by at least one of:

(i) determining an inverse hash representation thereof if said type is a first type; and

(ii) determining a decompressed representation thereof if said type is not a first type; and

(iii) retaining the encoded syntactic element.

According to another aspect of the invention there is provided an apparatus for parsing a markup language document comprising syntactic elements, said apparatus comprising:

identifying means for identifying a type of the element;

processing means for processing the element by determining a hash representation thereof if said type is a first type; and

augmenting means for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided an apparatus for validating a markup language document against a VRD, said apparatus comprising:

(a) means for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said means comprising:

(i) means for determining a hierarchy position of said document tag;

(ii) means for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) means for storing said extended hashed representation of said document tag if said document tag is more deeply nested than an extended hashed representation of a previous document tag;

(b) means for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said means comprising:

(i) means for determining a hierarchy position of said tag;

(ii) means for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) means for storing said extended hashed representation of said tag in a list; and

(c) means for establishing whether said extended hashed representation of said document tag is one of to be found in said list, and is a valid subset of a member of said list, thereby validating said markup language document.

According to another aspect of the invention there is provided an apparatus for validating a markup language document against a VRD, said apparatus comprising:

(a) means for processing said VRD, for each structural element identified therein, said means comprising:

(i) means for determining syntactic attributes of said structural element;

(ii) means for determining a hashed representation of said structural element; and

(iii) means for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) means for processing the markup language document, for each document structural element identified therein, said means comprising:

(i) means for determining syntactic attributes of said document structural element;

(ii) means for determining a hashed representation of said document structural element; and

(iii) means for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) means for comparing syntactic attributes and hashed representations of said each document structural element in the structural representation of the document to corresponding syntactic attributes and hashed representations in said structural representation of said VRD to thereby establish validity of the markup language document.

According to another aspect of the invention there is provided an apparatus for encoding a markup language document comprising syntactic elements, to form an at least partial structural representation of the document, said apparatus comprising:

means for identifying a type of the syntactic element; and

means for processing the syntactic element by one of:

(i) determining a hash representation thereof if said type is a first type;

(ii) determining a compressed representation thereof if said type is not a first type; and

(iii) retaining the syntactic element.

According to another aspect of the invention there is provided an apparatus for decoding a markup language document comprising encoded syntactic elements, said apparatus comprising:

means for identifying a type of the encoded syntactic element;

means for processing the encoded syntactic element by at least one of:

(i) determining an inverse hash representation thereof if said type is a first type;

(ii) determining a decompressed representation thereof if said type is not a first type; and

(iii) retaining the encoded syntactic element.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to parse a markup language document comprising syntactic elements, said program comprising:

code for identifying a type of an element;

code for processing the element by determining a hash representation thereof if said type is a first type; and

code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said document tag;

(ii) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) code for storing said extended hashed representation of said document tag if said tag is more deeply nested than a previous document tag;

(b) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said tag;

(ii) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) code for storing said extended hashed representation of said tag in a list; and

(c) code for validating said markup language document if said extended hashed representation of said document tag is one of found in said list, and is a valid subset of a member of said list.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing said VRD, for each structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said structural element;

(ii) code for determining a hashed representation of said structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said document structural element;

(ii) code for determining a hashed representation of said document structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) code for validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to encode a markup language document comprising syntactic elements, said program comprising:

code for identifying a type of the syntactic element; and

code for processing the syntactic element by one of:

(i) determining a hash representation thereof if said type is a first type;

(ii) determining a compressed representation thereof if said type is not a first type; and

(iii) retaining the syntactic element.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to decode a markup language document comprising encoded syntactic elements, said program comprising:

code for identifying a type of the encoded syntactic element;

code for processing the encoded syntactic element by at least one of:

(i) determining an inverse hash representation thereof if said type is a first type; and

(ii) determining a decompressed representation thereof if said type is not a first type; and

(iii) retaining the encoded syntactic element.

According to another aspect of the invention there is provided a computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to parse a markup language document, said program comprising:

code for identifying a type of the element;

code for processing the element by determining a hash representation thereof if said type is a first type; and

code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided a computer program product including a computer readable medium having recorded thereon a

computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said document tag;

(ii) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) code for storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(b) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said tag;

(ii) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) code for storing said extended hashed representation of said tag in a list; and

(c) code for validating said markup language document if said extended hashed representation of said document tag is one of found in said list and is a valid subset of a member of said list.

According to another aspect of the invention there is provided a computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing said VRD, for each structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said structural element;

(ii) code for determining a hashed representation of said structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said document structural element;

(ii) code for determining a hashed representation of said document structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) code for validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

According to another aspect of the invention there is provided an at least partial structural representation a markup language document comprising syntactic elements, said at least partial representation having been produced by a method comprising, for one of said syntactic elements, the steps of:

identifying a type of the element;

processing the element by determining a hash representation thereof if said type is a first type; and

augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided an apparatus for parsing a markup language document comprising syntactic elements, said apparatus comprising:

a processor;

a memory for storing (i) the document, and (ii) a program which is configured to make the processor execute a procedure to parse the document;

said program comprising:

(i) code for identifying a type of an element;

(ii) code for processing the element by determining a hash representation thereof if said type is a first type; and

(iii) code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided an apparatus for validating a markup language document comprising syntactic elements against a VRD comprising syntactic elements, said apparatus comprising:

(a) a processor;

(b) a memory for storing (i) the document, (ii) said VRD, and (iii) a program which is configured to make the processor execute a procedure to validate the document;

(c) said program comprising:

(ca) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(caa) code for determining a hierarchy position of said document tag;

(cab) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(cac) code for storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(cb) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said means comprising:

(cba) code for determining a hierarchy position of said tag;

(cbb) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(cbc) code for storing said extended hashed representation of said tag in a list; and

(cc) code for establishing whether said extended hashed representation of said document tag is one of to be found in said list, and is a valid subset of a member of said list, thereby validating said markup language document.

According to another aspect of the invention there is provided an apparatus for validating a markup language document containing syntactic elements against a VRD containing syntactic elements, said apparatus comprising:

(a) a processor;

(b) a memory for storing (i) the document, (ii) said VRD, and (iii) a program which is configured to make the processor execute a procedure to validate the document;

(c) said program comprising:

(ca) code for processing said VRD, for each structural element identified therein, said code comprising:

(caa) code for determining syntactic attributes of said structural element;

(cab) code for determining a hashed representation of said structural element; and

(cac) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(cb) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(caa) code for determining syntactic attributes of said document structural element;

(cab) code for determining a hashed representation of said document structural element; and

(cac) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(cc) code for comparing syntactic attributes and hashed representations of said each document structural element in the structural representation of the document to corresponding syntactic attributes and hashed representations in said structural representation of said VRD to thereby establish validity of the markup language document.

According to another aspect of the invention there is provided a method of validating a markup language document against a VRD, said method comprising steps of:

determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

storing said first extended hashed representation(s) in a VRD list;

determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

According to another aspect of the invention there is provided an apparatus for validating a markup language document against a VRD, said apparatus comprising:

means for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

means for storing said first extended hashed representation(s) in a VRD list;

means for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

means for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

code for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

code for storing said first extended hashed representation(s) in a VRD list;

code for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

code for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

According to another aspect of the invention there is provided a computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

code for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

code for storing said first extended hashed representation(s) in a VRD list;

code for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

code for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

Detailed Description including Best Mode

Where reference is made in any one or more of the accompanying drawings to steps and/or features, which have the same reference numerals, those steps and/or features have for the purposes of this description the same function(s) or operation(s), unless the contrary intention appears.

The inventive concept disclosed in this specification is based on the idea that memory requirements of an XML parser can be reduced, and various performance metrics can be improved, by performing a "perfect" hash of the XML tags, and possibly other elements within an XML file. A hash function is a function, mathematical or otherwise, that takes an input string, and converts it to an output code number called a hash value. A perfect hash function is one which creates a unique code number for a unique input string within a preset domain. The input string can be composed, for example, of alpha numeric

characters, or other characters approved by the World Wide Web Consortium, and must be less than a certain length dictated by the hash process specifics. Alternatively, or in addition, the input string can be constrained in other ways, for example in terms of a probability of code number collision based on input context. This idea allows an arbitrary XML tag to be treated as a numeral or code, which can be stored in numeric form in memory. Since a parser normally preserves some portion of an XML structure in memory as the structure is parsed, conversion of XML tags to unique numerals allows memory requirements to be reduced, and furthermore, allows string-to-string comparisons to be replaced with equivalent, but much faster numerical comparisons.

The principles of the arrangements described herein have general applicability to parsing documents using a wide variety of mark-up languages. For ease of explanation, the disclosed arrangements are described with reference to the XML language. This is not intended, however, to limit the scope of the inventive concept. For example, the disclosed arrangements can also be applied to a UTF-16 transformation format (see International Standard ISO/IEC 10646-1 for further details of UTF-16).

Figs. 2(a) and 2(b) depict a prior art SAX parser process 236, which supports optional well-formedness and/or validation checking sub-processes.

In Fig. 2(a), a mark-up document, in the present case an XML document, is opened in an initial step 200. Thereafter, a decision step 202 tests whether the document contains any unprocessed (ie unparsed) characters, and if this is the case, a character is read and stored in a string in a following step 204. If further characters are, however, not detected in the testing step 202, the parsing process 236 terminates in a step 234.

Following the step 204, a check is performed in a testing step 206 to determine whether a complete syntactic element has yet been assembled, and if so, the parser process 236 proceeds to a "Syntactic Type" identification step 210. If, on the other hand, a complete syntactic element has not yet been assembled, the parser process 236 is directed to a decision block 208 which determines if any further characters are available in the document. If additional characters are available, the parser process 236 is directed according to a "yes" arrow back to the step 204. Alternatively, if no more characters are

available, then the process 236 is directed in accordance with a "no" arrow to the syntactic element type identification step 210.

The "type identification" step 210 identifies a "type" for the assembled syntactic element, after which the element string is placed, in a step 212, into a memory representation of the document structure, thereby augmenting the representation as it has been assembled to this point. The memory representation of the document structure, which is typically, in the case of event driven parsers, a partial structural representation of the document, can be a simple list.

After the step 212, the process 236 is directed to a testing step 242, which determines whether a well-formedness check is to be performed. Well-formedness checks ensure that the document meets appropriate "well-formedness constraints", as defined on page 5 of "Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation, 6 October 2000", which is available on the Internet at http:\\www.w3.org\tr\2000\rec-xml-20001006.html. Well-formedness checks test the document for compliance with general structure rules, particularly whether tags in a document have been properly nested. If such a check is to be performed, then the process 236 is directed in accordance with a "yes" arrow to "a" on a dashed boundary line 246. The dashed boundary line 246, along with reference letters "a" to "d" is mirrored by a corresponding boundary line in Fig. 2(b), in relation to which the process 236 is further described. If the well-formedness check is not to be performed, then the process 236 is directed in accordance with a "no" arrow from the testing step 242 to a testing step 244 which determines whether a "validation check" is to be performed. Validation checks involve a comparison of syntactic elements in a document against validity constraints defined in a Validation Reference Document (referred to as a VRD for the sake of brevity) such as a document type definition (DTD), as described in Section 5.1 of the aforementioned W3C Recommendation. DTDs and XML Schemas are examples of VRDs against which validation checks can be performed, however validation checks as described herein can be performed against other types of VRDs. This comparison procedure verifies correct syntactic placement of elements to a greater extent than the mere well-formedness check. If the validation check is to be performed, then the process

236 is directed in accordance with a "yes" arrow to "b" on the dashed boundary line 246. If, on the other hand, the validation check is not to be performed, then the process 236 is directed in accordance with a "no" arrow to "c" on the dashed boundary line 246.

If the well-formedness check is elected, then the process 236 is directed from "a" on the boundary line 246 to an optional sub-process 238, and in particular to a well-formedness checking step 214 found therein. The optional nature of the process 238 is denoted by the dashed rectangle outline thereof. If the validity check is elected, then the process 236 is directed from "b" on the boundary line 246 to an optional sub-process 240, and in particular to a validity checking step 220 found therein. The optional nature of the process 240 is denoted by the dashed rectangle outline thereof. If the validity check is not elected, then the process 236 is directed from "c" on the boundary line 246 to an action selection step 226.

If the well-formedness check is elected, then after the well-formedness step 214, if an error is detected in the following error checking step 216, corrective action and/or error indication takes place as indicated by an arrow 218. If, on the other hand, no errors are detected, then the parser process 236 is directed from the step 216 to the sub-process 240, in which the validation check is performed in the step 220. As noted, the parsing processing 236 can be directed to the validation checking step 220 either from the error checking step 216, or alternatively, the well-formedness checking sub-process 238 can be by-passed, and the process 236 can be directed directly to the validation checking step 220 from "b" on the boundary line 246. The optional well-formedness sub-process 238 can be bypassed if the appropriate decisions are made in the testing steps 242 and 244 (see Fig. 2(a)).

As noted, the validation checking step 220 involves a comparison of the identified syntactic element in the markup document being considered against a document type definition (DTD). This comparison procedure verifies correct syntactic placement of elements to a greater extent than the mere well-formedness check described in relation to the sub-process 238.

Following the validation step 220, if an error is detected in an error checking step 222, corrective action is taken, and/or an error indication is produced, as depicted by an

arrow 224. Alternatively, if no error is detected, the parser process 236 proceeds to the action selection step 226, where an action is selected based upon the type of the syntactic element being considered. The optional sub-processes 238 and 240 can both be bypassed, if the appropriate decisions are made at the decision steps 242 and 244 in Fig. 2(a). If both of the aforementioned sub-processes are bypassed, then as noted the parsing process 236 is directed from "c" on the boundary line 246 directly to the action selection step 226.

If the syntactic element is a tag, then as depicted by an arrow 228 the tag value, or a representative string, is sent to the application in respect of which the parsing process is being performed, and a memory representation of the tag is maintained. If, on the other hand, the element type is a non-tag type, then as depicted by an arrow 230, the element value string is sent to the associated application, and the memory representation of the element is deleted. Finally, the parsing process 236 is directed, as depicted by an arrow 232, to "d" on the dashed boundary line in Fig. 2(b), and from "d" on the corresponding dashed boundary line 246 in Fig. 2(a) to the character testing step 202.

Significant memory requirements arise from the verbose nature of the XML document, resulting in correspondingly significant memory requirements to store the document structure in its original string form. This document structure is referred to in the step 212. Furthermore, an associated significant processing load, relating to performance of string comparisons between variable length alpha-numeric strings, arises both in the well-formedness checking step 214, and in the validation checking step 220.

A partial memory representation of the document must typically be stored, and string checking must typically be performed, both (i) in relation to the step 214 in regard to checking for closure of hierarchy branches, namely matching end tags to start tags, and also for checking for non-overlapping branches, and (ii) in relation to the step 220, in which similar processes are required as in (i), as well as checking conformity of structure and tag names against the DTD.

A parser must normally preserve some portion of an XML structure in memory as the XML structure is parsed. Even for a SAX parser, a local portion of the XML structure must be retained in memory for correct operation. If however each XML tag is converted to a unique numeral using a hash function, memory requirements are typically

reduced, since the numeral resulting from the hash operation is smaller than the associated arbitrary-length XML tag string. Furthermore, string-to-string comparisons, required for matching beginning & end tags, can be replaced with much faster numerical comparisons, thereby reducing the processing load.

Typical hash algorithms include (i) Cyclic Redundancy Coding (CRC) algorithms (commonly used for signature analysis or error-detection/correction in data transfer & storage), (ii) fully lossless encoding algorithms, and (iii) Huffman encoding algorithms.

Typically, a suitable hash algorithm must be static in its operation, or in other words it must always return the same hash result for the identical input conditions over the required set of data. The required set of data can, however, vary according to the circumstance. The data set can thus typically comprise at least an entire markup document, but can also include a relevant DTD or XML Schema, linked markup documents, and related or linked markup documents in different languages, eg a CSS document referenced by an XML document. A static hash algorithm can, however, be used where necessary by resetting the algorithm whenever tag syntax is encountered, for example whenever the non-literal '<' character is found in an XML document. The hash algorithm can also be reset where an <!ELEMENT string is found in an XML DTD document, or wherever a valid tag selector is permitted in a CSS document.

A reference in an input markup document can be used to signal, or to select a suitable hash algorithm. This can be done in much the same way as markup documents can reference other markup documents, DTDs, stylesheets, character encodings, namespaces, and so on. For example a particular hash algorithm can be identified with a particular namespace, thereby permitting indirect reference to a hash algorithm via a namespace reference within a document. A hash algorithm implementation can be wholly, or partially included within a markup document, along with associated parameterisation. Such methods of referencing or including hash algorithms can be useful for optimisation purposes, where different hash methods have been optimised for use with particular markup documents, thereby improving performance and memory usage in destination devices or systems. Alternatively, the aforementioned referencing methods can be useful

for matching purposes. This refers to applications involving one or more markup documents, where error checking or completion of parsing or other functions are required, and where one or more other documents (e.g. a DTD) have already been hashed by the same algorithm.

Further refinements are possible in the above approach, for example involving optional hashing of DTDs. This reduces Read Only Memory (ROM) requirements for storing DTDs, and provides for faster validation processing of XML documents, by allowing comparison of numerical values rather than (slower) string comparisons.

Figs. 3(a), 3(b) and 3(c) illustrate one arrangement of an improved SAX parser process 344. In Fig. 3(a), steps 300, to 310 are identical to corresponding steps 200 to 210 which have been described in relation to Fig. 2(a). After the step 310, the assembled syntactic element is tested to ascertain its nature as a tag, or another element type, in a testing step 312. If the element is a tag, the parsing process 344 is directed to a hash step 318 by an arrow 316. The hash step 318 determines, using respective processors 414 or 505 in Figs. 5 and 6, a unique numeric representation of the syntactic element. This results in a more memory efficient representation of the element, which also lends itself to simpler and faster comparison operations in the numeric, rather than the alpha-numeric domain. Both the element string depicting the syntactic element, and the hash value thereof, are retained at this point of the process 344, however it is the hash value, and not the string value, which is inserted, in the step 318, into the memory representation of the document structure using respective memories 418 and 506 (see Figs. 5 and 6).

In order to better appreciate operation of the parsing process 344 as described in relation to Figs. 3(a), 3(b) and 3(c), parsing of the exemplary XML fragment [1] is considered firstly in relation to the parsing process 236 described in relation to Fig. 2. In this case, the XML fragment [1] yields the following hierarchical representation of parsed mark-up tags in the sub-process 212:

```
205  Shakespeare
215       div
220         mult
221         /mult
225       banquo
235                quote
240              quote                                    [2]
245       /banquo
250         Hamlet
251                    quote
252                  /quote
253         /Hamlet
255  /Shakespeare
```

In contrast, the differentiated treatment of tag elements and non-tag elements in the parsing process 344, as described in relation to Fig. 3(a), results in an equivalent hierarchical representation being generated by the step 318. The equivalent hierarchical representation is depicted in [3]. The hierarchical representation in [3] is made up of parsed hashed mark-up tags. For the sake of this example, a domain of tag names is constrained to those shown in the following Table 1, and a hash mapping (which is functionally equivalent to application of a hash "function") is shown in the following table:

Tag	Hash Code Number
Shakespeare	133
Div	326
Mult	371
Banquo	787
Quote	629
Hamlet	411

Table 1. Hash Mapping

Based on the above hash mapping, the following hierarchical representation of the XML fragment shown in [1] results:

```
205  133
215          326
222           371
223           /371
225          787
235                  629
240                 /629                                          [3]
245          /787
254           411
255                      629
256                     /629
257           /411
255  /133
```

Returning to Fig. 3(a), the parsing process 344 is directed from the step 314 to "a" on a dashed boundary line 356. The dashed boundary line 356, along with reference letters "a" and "b" is mirrored by a corresponding boundary line in Fig. 3(b), in relation to which the process 344 is further described.

Turning to Fig. 3(b), the process 344 continues from "a" on the dashed boundary line 356 to a testing step 350 which determines whether a well-formedness check is to be performed. If such a check is to be performed, then the process 344 is directed in accordance with a "yes" arrow to "c" on the boundary line 358. The dashed boundary line 358, along with reference letters "c" to "f" is mirrored by a corresponding boundary line in Fig. 3(c), in relation to which the process 344 is further described. If the well-formedness check is not to be performed, then the process 344 is directed in accordance with a "no" arrow to a testing step 352 which determines whether a validation check is to be performed. If the validation check is to be performed, then the process 344 is directed in accordance with a "yes" arrow to "e" on the dashed boundary line 358. If, on the other hand, the validation check is not to be performed, then the process 344 is directed to "d" on the dashed boundary line 358.

Turning to Fig. 3(c) if the well-formedness check is to be performed, then the process 344 is directed from "c" on the dashed boundary line 358 to a well-formedness checking step 320. If, on the other hand, the well-formedness check is not elected, then the process 344 is directed from "e" on the dashed boundary line 358 to a validation checking step 326. If neither a well-formedness check or validation check is elected, then

the process 344 is directed from "d" on the dashed boundary line 358 to an action selection step 334.

The well-formedness checking step 320 performs well-formedness checking using respective processors 414 or 505 and forms part of an optional process 346. The optional nature of the process 346 is depicted by use of dashed lines. Similarly, the validation step 326 forms part of an optional sub-process 348, the optional nature thereof being depicted by use of dashed lines.

It is apparent that the hierarchical representation depicted in [3] allows string comparisons to be replaced by faster and more efficient numerical comparisons, thereby reducing the associated computational burden. Furthermore, the hierarchical representation shown in [3] is a more memory-efficient representation, than that shown in [1] and accordingly the representation shown in [3] is more suited to memory-constrained applications as previously discussed.

Returning to Fig. 3(c), if well-formedness checking is elected, then after well-formedness checking is performed in the step 320, the parsing process 344 is directed to an error checking step 322, whereupon if an error is detected, as depicted by an arrow 324, corrective action is taken, and/or an error is indicated. The well-formedness check typically considers whether tags in a document have been properly nested. Thus, for example, having reference to [2] the tag pair "Hamlet" and "/Hamlet" are properly nested within the tag pair "Shakespeare" and "/Shakespeare" since the "Hamlet" tag pair is fully nested within the "Shakespeare" tag pair, and the tag pairs do not, for example, overlap each other.

If, on the other hand, no error is detected, the parsing process 344 is directed to the optional process 348, in which the validation checking step 326, using respective processors 414 or 505, is performed with reference to a DTD or an XML Schema. As noted, validation checking is a more detailed form of checking than well-formedness checking. Thus, for example, whereas the well-formedness check considers whether the "Hamlet" tag pair is properly nested within the "Shakespeare" tag pair, validity checking, in contrast, both checks for proper nesting in the sense that the "Hamlet" tag pair is fully nested within the "Shakespeare" tag pair, but also checks whether "Hamlet" tag pairs may

legally be nested in this way. There may, for example, be a situation where, in fact, "Shakespeare" tag pairs must be nested within "Hamlet" tag pairs, rather than the other way around. Thus, the validity checking process checks hierarchical relationships of tags, in this case being whether "Hamlet" tag pairs may be nested within "Shakespeare" tag pairs, as well as considering whether nesting has been properly, namely completely, performed.

In order to perform the validation step 326, DTD or XML Schema tags are first hashed in a hashing step 328, in order to bring the DTD/XML Schema memory representation into conformity with the hashed nature of the mark-up document which has been generated by the hash step 318. The validation checking step 326 compares the mark-up document structural representation generated in the step 318 to the structural representation of the DTD/XML Schema generated in the step 328, to verify correct syntactic placement of syntactic elements in the markup document, noting that the string comparisons required for this comparison as used in step 220 in relation to Fig. 2, are now replaced, in Fig. 3(c), by faster and more efficient numerical comparisons, as a result of the hashing operations in steps 328 and 318.

After validation, the process 344 is directed to an error checking step 330, in which corrective action and/or error indication is performed as indicated by an arrow 332. If no errors are detected, the parsing process 344 proceeds to an action selection step 334, whereupon if the syntactic element is a tag type, the corresponding tag string is sent to the application in respect of which the parsing process is being performed, and the tag string itself is deleted from memory, this being either 418 or 512 in Figs. 5 and 6 respectively. The associated hashed tag memory representation is, however, retained. Accordingly, no string-based memory representation of the tag is retained, other than one copy of the currently parsed tag string. The memory representation of the tag is thus only in hashed form. If the element syntactic type is either a non-tag type, or a non-tag name type, then as depicted by an arrow 338, the value of the element, or a string representation thereof is sent to the associated application, and the associated memory representation is deleted. The parsing process 344 now loops back, as indicated by an arrow 340, to "f" on the dashed boundary line 358, and thereafter to the corresponding "f" on the dashed boundary

line 358 in Fig. 3(b), and thereafter to "b" on the dashed boundary line 356, and thereafter to "b" on the dashed boundary line 356 in Fig. 3(a), and finally to the character testing step 302. If no further characters are detected, the parsing process 344 terminates in a step 342.

The XML document fragment [1], with tags in hashed form, has the following form:

```
505 <133>
110 <!--This is a comment-->
515 <326 class="preface" Name1="value1" name2="value2">
520 <371 list=&lt;> </371>
525 <787>
130 Say                                                    [4]
535 <629>
540 goodnight  </629>,
545 Hamlet.</787>
550 <411><629>Goodnight, Hamlet. </629></411>
555 </133>
```

The representation of closing tags (which typically use syntax: </section> as opposed to start tags which use syntax <section>) can be defined in various ways, thereby attaining more, or less, compatibility with the XML standard. It is noted that start tags and end tags are considered, in the present description, to be "equivalent types". Furthermore, the fact that the start and end tag perform a collective function, namely delimiting sections of document content, is taken to mean that there is a relationship between the two tags. It is further noted that the aforementioned syntax for start and end tags means that the end tag is a modification of the start tag, wherein a distinguishing character, namely a "/" is incorporated into the start tag in order to produce a corresponding end tag. Compatibility with the XML standard can be more important in some instances than in others. In the preferred embodiment, the '/' character of a current tag string is typically removed prior to hashing the following tag name, in order that identical start and end tag names return the same numeral from the hashing function. An XML tag is exemplified by </Name Attribute>, as can be seen in Section 3.1 of "Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation, 6 October 2000", which is available on the Internet at http:\\www.w3.org\tr\2000\rec-xml-

20001006.html). In the present description, the term "tag" can refer, depending on the context, to either a part of, or the entirety of, the particular tag being considered. Alternatively, there may be situations where it is desired to retain an equivalent representation of the '/' character (identifying the end tag) in memory. This can be done in a variety of ways, such as: (i) reinserting the '/' character or an equivalent character into memory in proximity to the end tag hash numeral so as to indicate that it is an end tag, (ii) using a boolean value to indicate the end or start state of a hashed tag, or (iii) negating the end tag hashed value so that a simple addition of start and end tags yields zero for a perfect match. In Option (iii), the hashed start tag has been modified by an operator, in the present case a simple negation operation, in order to produce the requisite hashed end tag. Option (iii) requires that a sign bit be guaranteed to be free from influence by the hashing algorithm. This option is, in fact, very similar to the boolean flag option (ii).

Furthermore, structured hash numbers can be generated in which a hash number for a nested tag can explicitly indicate the higher-level XML tags within which the first tag is nested. Thus, for example, where tag 123 is nested inside tag 987, then instead of being designated as nested tag 123, it can be designated as 987.123. This structured, or "extended", hashing can allow further parsing performance improvements by reducing structure-spanning operations, ie by reducing an amount of the XML document which must be held in memory while the end-points of a tag pair are being searched for.

It is also noted that extended representations need not be based upon hashing, but can also be based upon strings, or "enumeration", which is a process whereby a mapping is defined between tag names and numerals, thereby creating an enumeration table or index. A simple form of enumeration is to merely list all the tag names, and to number the listed tags. Thus, for example, a concatenated string of the form "Shakespeare.banquo.quote" represents a string-based extended representation of three concatenated tags.

A structured equivalent hashed markup example for the XML fragment [3] is presented in [5] below using negated, hashed end tags.

```
133
133.326
133.371
133.-371
133.787
133.787.629          -> 013307870629
133.787.-629     -> -013307870629
133.-787                                                    [5]
133.411
133.411.629
133.411.-629
133.-411
-133
```

In [5], the structure of nested tags is converted from the form shown in [3] into a series of concatenated hashed tags, in which each subsequent lower (ie. more deeply nested) hierarchical level of hashed tag is directly linked to its previous upper hierarchical levels. This allows simple numerical comparison to be performed with a similarly parsed structure from a hashed DTD. In fact, each line in [5] is represented, as shown in [5] for lines 4 and 5, by a single numeral which is combined by concatenation of the set of hashed tags encountered. This single numeral represents in a very compressed form both the identities and relationships of the original input tags, and accordingly enables a very efficient comparison method with a similarly hashed DTD. It can be seen that the numerical tag sets can be used to represent the document structure in a highly compressed form. A validation check can be performed using merely the hashed start tag sets, noting that each such set represents the deepest, and entire, structure of each branch of the document structure. For instance, the structure of [5] can be minimally represented in [6] as follows:

```
01330326
01330371                                                    [6]
013307870629
013304110629
```

A DTD or XML schema structure can also be represented by the same method.

A single, or multiple set of numerical comparisons between a tag set from the parsed & hashed input document and a tag set from the parsed & hashed DTD replaces a series of string and structure comparisons normally required in XML parser validation. It

can be recognized that any alternate valid structures defined by a DTD or XML schema can be encoded into unique hashed tag set numerals for later comparison with hashed tag set numerals generated from an input XML document.

Fig. 4 depicts a process 600 for validating a mark-up document against a VRD such as, for example, a DTD, or an XML schema. The process 600 commences with a step 602 in which a markup document to be validated is opened. Therafter, in a step 604, a current extended tag is reset by a respective processor 414 or 505 in Fig. 5 or Fig. 6. In the description relating to Fig. 4, the terms "tag", "extended tag", "temporary tag" and so on refer to the hashed representations of the respective tags. In a following step 606, a temporary tag root is reset by one of the respective processors 414 or 505, after which a next tag in the markup document is identified in a step 608. Thereafter, a testing step 610 determines whether the tag identified in the step 608 is a start tag, in which event the process 600 is directed in accordance with a "yes" arrow to a step 612 which adds the tag identified in the step 608 to the extended tag using one of the respective processors 414 or 505. The process 600 is then directed from the step 612 back to the step 608.

If the testing step 610 determines that the next tag is not a start tag, then the process 600 is directed in accordance with a "no" arrow to a testing step 614, which determined whether the extended tag = "0", which represents the root level of the document. If the "0" value is detected, then the process 600 is directed in accordance with a "yes" arrow to a testing step 624 which determines whether the end of the document has been reached. If this is not the case, then the process is directed in accordance with a "no" arrow back to the step 606. It is noted that detection of a "0" value in the step 614 may also result from an document structure which is not well formed, such as would be the case for a structure having a mismatched number of start and end tags.

If the testing step 614 determines that the extended tag value is not equal to "0", then the process is directed in accordance with a "no" arrow to a testing step 616, which determines whether the extended tag is equal to the temp tag root value. If this is found not to be the case, then the process 600 is directed in accordance with a "no" arrow to a step 618 which stores the extended tag in a document list in a respective memory 418 or

506. If on the other hand, the testing step 616 determines that the extended tag is equal to the temp tag root, then the process 600 is directed in accordance with a "yes" arrow to a step 620 which removes the lowest (namely the most deeply nested) tag from the extended tag, using a respective processor 414 or 505. Thereafter, in a step 622, the process 600 copies the extended tag to the temp tag root, after which the process is directed back to the step 608.

Prior to returning to the testing step 624, it is noted that the process 600 as heretofore described is directed to the markup document whose validity is being checked. There is also, however, an identical process, not explicitly described, which is applied to the validation reference document (VRD) to thereby produce a VRD list against which the document list produced by the process 600 can be tested. The process 600 and the equivalent process directed to the VRD typically occur at different times. The process 600 occurs for every document being validated, and produces a list of extended hash representations for each particular document being validated. The VRD list can be produced substantially concurrently with the process 600, providing that the VRD list is completed prior to the step 626. Alternately, the VRD process can be performed off line, and the resultant list provided to the process 600 prior to the step 626.

Returning to the step 626, and since the VRD list is available as noted, the step 626 determines whether every entry in the document list is to be found in the VRD list. If this is the case, then the process 600 is directed in accordance with a "yes" arrow to a step 628 which declares that the document is not detected as invalid. If on the other hand, the document list has an entry which is not to be found in the VRD list, then the process 600 is directed in accordance with a "no" arrow to a step 630 which declares that the document is invalid.

The above description compares, as described in more detail in regard to the step 626, all the document list entries with all the VRD list entries. An alternate process is to test each extended tag, after the step 616, against the complete VRD list in a step similar to the step 626, in which event if the document extended tag is not to be found in the VRD list, the process 600 can proceed directly to the step 630, saving unnecessary further testing. In the event that the extended tag is however to be found, then the process 600

can be directed to the step 620 and so on. The alternate arrangement provides earlier recognition of an error, and immediately aborts the validation process, which provides added efficiency provided the VRD list is relatively short. If a complete validation check is implemented with the above method, then the step 628 indicates that the document being considered is valid.

In order to further illustrate the validation method described, the following structure fragment is considered, in which start tags are "01" to "05", and the corresponding end tags are "-01" to "-05" respectively.

```
01
    02
        03
        -03
        04                                                    [7]
        -04
    -02
    05
    -05
-01
```

In functional terms, the process 600 traverses down to the deepest part of a branch in the hierarchical structure of the mark-up document, namely from "01" on the first line of [7] to "03" on the third line of [7], and stores an extended hash representation for the deepest part of that particular branch, namely "010203". The process then traverses up the branch, discarding end tags, until it finds another start tag which indicates a new branch to pursue, which is "04" on line 5 of [7] in this example. As the process traverses down the new branch, the process preserves the extended hash representations of higher levels of the hierarchy, until it has stepped back above those levels. An error in the document structure, resulting in an invalid document or a document which is not well formed, will typically return extended hash representations that do not match those of the VRD. The step 620 may optionally include well-formedness checks of the retrieved end tag against the previous start tag, thereby providing a well-formedness match if the document is well formed. It is noted that the previous start tag is the lowest tag in the extended hash representation. For example the DTD/XML Schema may return end tags

at the testing step 610 that do not match the lowest start tag in the extended hashed representation in the step 620, thereby failing a well-formedness check.

The test at the step 626 typically seeks to match the extended hashed representations of the mark-up document structure against those hashed representations listed for the DTD of that document. The hashed representations of the document structure will typically be a subset of the deepest structural representations from the DTD list. Accordingly, a valid XML document is permitted to contain any legal subset of the structural nesting defined in the corresponding VRD, or DTD. Therefore, a typical test in the step 626 includes comparisons of shallower hashed structural representations of the document against a deeper hashed representation of the DTD. Thus, for example, an extended hashed representation "0123" from the XML document would be assessed as "valid" when compared to a hashed representation "01230456" from the corresponding DTD.

The validation process 600 shown in Fig. 4 can be optimized to check the more complex parts of a document structure, such as the most deeply nested portions, in a fast but incomplete check of validity. Thus, an optimal combination of speed and validity sensitivity can be selected, in order to implement a particular validating parser having arbitrary performance characteristics.

The validation method 600 can also be modified to perform at least portions of "standard" well-formedness checks. Thus, for example, in the step 620, the hashed representation of the end tag can be checked against the lowest hierarchical hashed tag representation within the extended tag representation. If the aforementioned representations do not properly resolve to the same original tag identity, then the document is not well-formed, and a recovery, or error action can be performed.

The above method can be extended to include hashed representations of defined attributes within a structure, either separately, or together with structure checking.

It is apparent that this method of validation and well-formedness checking can be applied to an input document in a separate process to the process for parsing of the document structure and content. Thus, for example, the method 600 can be optimised in order to achieve an efficient and high-speed validity and well-formedness check that can

be performed even in environments where central processing unit (CPU) cycles and memory size are not particularly subject to major constraints. The advantage of performing a separate check in such systems includes the fact that a highly optimized check can be used to quickly discard "invalid" documents. This can save considerable time and processing of at least part of an invalid document, thereby preventing, for example, (i) parsing of the document into a full DOM tree representation and then performing validation checking only to find that the document is invalid, or (ii) commencing further processing of a first (valid) part of a document prior to detecting an invalid second part of the document, the further processing of the first part of the document being thereby rendered futile. Another advantage is that after a document is discovered to be invalid using the fast validation check, processing of a following job can be immediately commenced.

An "imperfect" hash process ie a hash process which is not guaranteed to produce a unique numeral for each alphanumeric input string, can be adequate in certain cases, in particular where the maximum length of XML tag strings is constrained, or is at least constrained to some level of probability. Furthermore, in cases where the set of XML tag strings is constrained to some limited number of character permutations, or is constrained with some probability to a limited number of character permutations, the imperfect hash process can be designed, or selected, to operate adequately.

A communications standard, or alternative public or private format(s) for numerical representation of a document structure can be defined or described based on the use of a hash algorithm. This technique allows a form of compression, which can be of benefit in transmission of XML data which normally involves transmission of a significant amount of data because of its verbosity, and human-readable ASCII form. Various options exist for retaining or discarding human-readability, for example by combining (perfect) hashing with other forms of compression, which are respectively applied to differing element types within an XML file. For instance, it is possible to replace XML string tags with unique, human-readable numerals derived from a perfect hashing algorithm. Un-hashed syntactic and other elements can also be compressed by a

lossless compression technique for transmission between processes or devices, thereby reducing the amount of transmitted data.

An inverse or reversible hash algorithm can be referenced or included where required as discussed in the previous paragraph. This is used where, for example, such an algorithm is needed to decode or decrypt one or more markup tags into a human-readable string for display or labelling purposes from a pre-hashed, transmitted markup document, where it is otherwise not necessary to do so for parsing and error-checking purposes. Another use of a reverse or inverse hash algorithm is to allow decryption of markup tags or other data to enable a restricted function or feature relating to the transmitted markup document. Reverse or inverse algorithms can also be used for matching a transmitter and a receiver of markup documents, where the reverse or inverse hash algorithm is already included in the receiver, and is not transmitted, but might be referenced in the markup document. Examples of reversible or invertible hash algorithms include (i) fully lossless encoding algorithms and (ii) Huffman encoding algorithms.

The aforementioned arrangements can be applied to any markup language, with particular advantages where one or more of the following conditions apply, namely (i) the markup language allows definition of tag names (e.g. XML, DTD, CSS, XSL, etc), (ii) tag names use large character encoding tables (e.g. UTF-16) and/or tag name length is not typically shorter than the hashed representation thereof, (iii) the intended application using or receiving a markup document typically requires representation of complex structures with more than one hierarchical level of nesting within a markup document, XML Schema, or DTD, (iv) some form of checking, typically well-formedness or validation, is required for the input markup document, (v) the markup parser and/or application have strong limitations on memory capacity (for example, embedded or low-cost CPU systems) or memory management (for example in systems having no virtual memory, or no dynamic memory allocation), and (vi) the markup parser and/or application need to operate quickly on potentially complex, highly-nested, markup documents.

The disclosed method of parsing a markup language document can be implemented in dedicated hardware such as one or more integrated circuits performing

the functions or sub functions of parsing a markup language document. Such dedicated hardware may include graphic processors, digital signal processors, or one or more microprocessors and associated memories.

The method of parsing a markup language document can alternatively be practiced using a special purpose embedded computer system 400, such as that shown in Fig. 5 wherein the processes of Figs. 3(a), 3(b), 3(c), and 4 may be implemented as software, such as an application program executing within the embedded computer system 400. The computer system 400 is typically integrated (embedded) into an end system such as a printer (not shown) and drives a printer engine 402 in the printer. In particular, the steps of the method of parsing a markup language are effected by instructions in the software that are carried out by the embedded computer. The software may be stored in a computer readable medium, including Read Only Memory (ROM) 418 or Random Access Memory (RAM) 418 or other types of memory (not shown). The software is loaded into the embedded computer during manufacture, or by software upgrades performed on-site.

The embedded computer system 400 comprises a computer module 410, input devices such as a switch module 422 for parameter setting, an output device such as a Liquid Crystal Display (LCD) showing job status, and the printer engine 402. The embedded computer 400 is typically physically integrated into the printer (not shown). Print jobs which originate at other computers (not shown) attached to a computer network 406 are sent to the embedded computer 400 by a connection 404 to an Input/Output (I/O) interface 408.

The embedded computer module 410 typically includes a processor unit 414, a memory unit 418, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), input/output (I/O) interfaces including a switch module and LCD interface 416, and an I/O interface 408 for the printer engine 402 and network 406. The components 408, and 414 to 418 of the embedded computer 410 typically communicate via an interconnected bus 412 and in a manner which results in a conventional mode of operation of the embedded computer system 410 known to those in

the relevant art. Typically, the program of the arrangement is resident in memory 418, and is read and controlled in its execution by the processor 414.

The method of parsing a markup language document can also be practiced using a conventional general-purpose computer system 500, such as that shown in Fig. 6 wherein the processes of Figs. 3(a), 3(b), 3(c), and 4 may be implemented as software, such as an application program executing within the computer system 500. This application is useful, for example, when hashing is used as a communication standard across a network between computers. Fig. 6 shows only one of the communicating computers being considered.

In particular, the steps of the method of parsing a markup language document are effected by instructions in the software that are carried out by the computer. The software may be divided into two separate parts, namely one part for carrying out the parsing methods, and another part to manage the user interface between the latter and the user. The software may be stored in a computer readable medium, including the storage devices described below, for example. The software is loaded into the computer from the computer readable medium, and then executed by the computer. A computer readable medium having such software or computer program recorded on it is a computer program product. The use of the computer program product in the computer preferably effects an advantageous apparatus for parsing a markup language document in accordance with the embodiments of the invention.

The computer system 500 comprises a computer module 501, input devices such as a keyboard 502 and mouse 503, output devices including a printer 515 and a display device 514. A Modulator-Demodulator (Modem) transceiver device 516 is used by the computer module 501 for communicating to and from a communications network 520, for example connectable via a telephone line 521 or other functional medium. The modem 516 can be used to obtain access to the Internet, other network systems, such as a Local Area Network (LAN) or a Wide Area Network (WAN), and the other personal computer (PC) 522 with which the computer 500 is communicating..

The computer module 501 typically includes at least one processor unit 505, a memory unit 506, for example formed from semiconductor random access memory

(RAM) and read only memory (ROM), input/output (I/O) interfaces including a video interface 507, and an I/O interface 513 for the keyboard 502 and mouse 503 and optionally a joystick (not illustrated), and an interface 508 for the modem 516.

A storage device 509 is provided and typically includes a hard disk drive 510 and a floppy disk drive 511. A magnetic tape drive (not illustrated) may also be used. A CD-ROM drive 512 is typically provided as a non-volatile source of data. The components 505 to 513 of the computer module 501, typically communicate via an interconnected bus 504 and in a manner which results in a conventional mode of operation of the computer system 500 known to those in the relevant art. Examples of computers on which the embodiments can be practised include IBM-PC's and compatibles, Sun Sparcstations or alike computer systems evolved therefrom.

Typically, the application program of the embodiment is resident on the hard disk drive 510, and is read and controlled in its execution by the processor 505. Intermediate storage of the program and any data fetched from the network 520 may be accomplished using the semiconductor memory 506, possibly in concert with the hard disk drive 510. In some instances, the application program may be supplied to the user encoded on a CD-ROM or floppy disk and read via the corresponding drive 512 or 511, or alternatively may be read by the user from the PC 522 over the network 520 via the modem device 516.

Still further, the software can also be loaded into the computer system 500 from other computer readable medium including magnetic tape, a ROM or integrated circuit, a magneto-optical disk, a radio or infra-red transmission channel between the computer module 501 and another device, a computer readable card such as a PCMCIA card, and the Internet and Intranets including email transmissions and information recorded on websites and the like. The foregoing is merely exemplary of relevant computer readable mediums. Other computer readable mediums may be practiced without departing from the scope and spirit of the invention.

Industrial Applicability

It is apparent from the above that the embodiment(s) of the invention are applicable to the computer and data processing industries.

The foregoing describes only some embodiments of the present invention, and modifications and/or changes can be made thereto without departing from the scope and spirit of the invention, the embodiments being illustrative and not restrictive.

Brief Description of the Drawings

A number of preferred embodiments of the present invention will now be described with reference to the drawings, in which:

Figs. 1(a) and 1(b) shows block representations of XML parser systems in which embodiments of the present invention can be practiced;

Figs. 2(a) and 2(b) depict a flow chart of method steps for a prior art SAX parser, including optional well-formedness and/or validation checking steps;

Figs. 3(a), 3(b) and 3(c) show an improved arrangement of the SAX parser of Figs. 2(a) and 2(b);

Fig. 4 depicts a process for validating a document against a reference document such as a DTD, or an XML schema.

Fig. 5 is a schematic block diagram of a special purpose embedded computer upon which an arrangement of the improved SAX parser can be practiced; and

Fig. 6 is a general purpose computer upon which an arrangement of the improved SAX parser can be practiced.

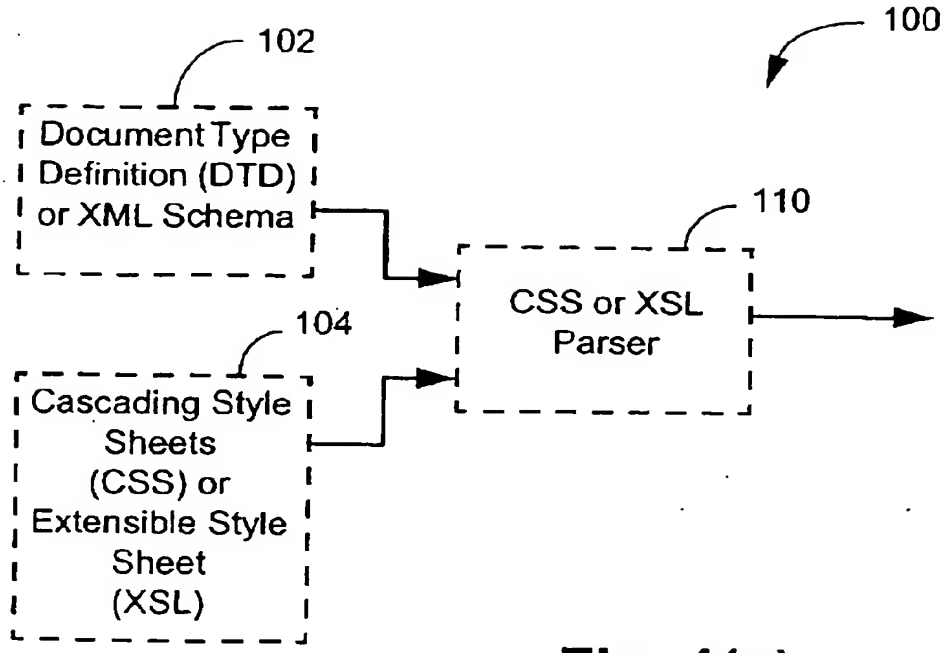

100
102
Document Type Definition (DTD) or XML Schema
110
CSS or XSL Parser
104
Cascading Style Sheets (CSS) or Extensible Style Sheet (XSL)

Fig. 1(a)

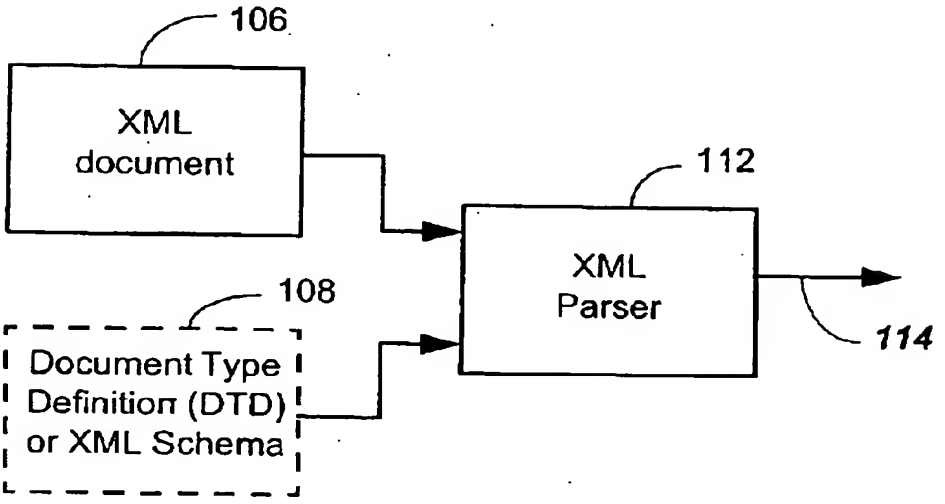

106
XML document
112
XML Parser
114
108
Document Type Definition (DTD) or XML Schema

Fig. 1(b)

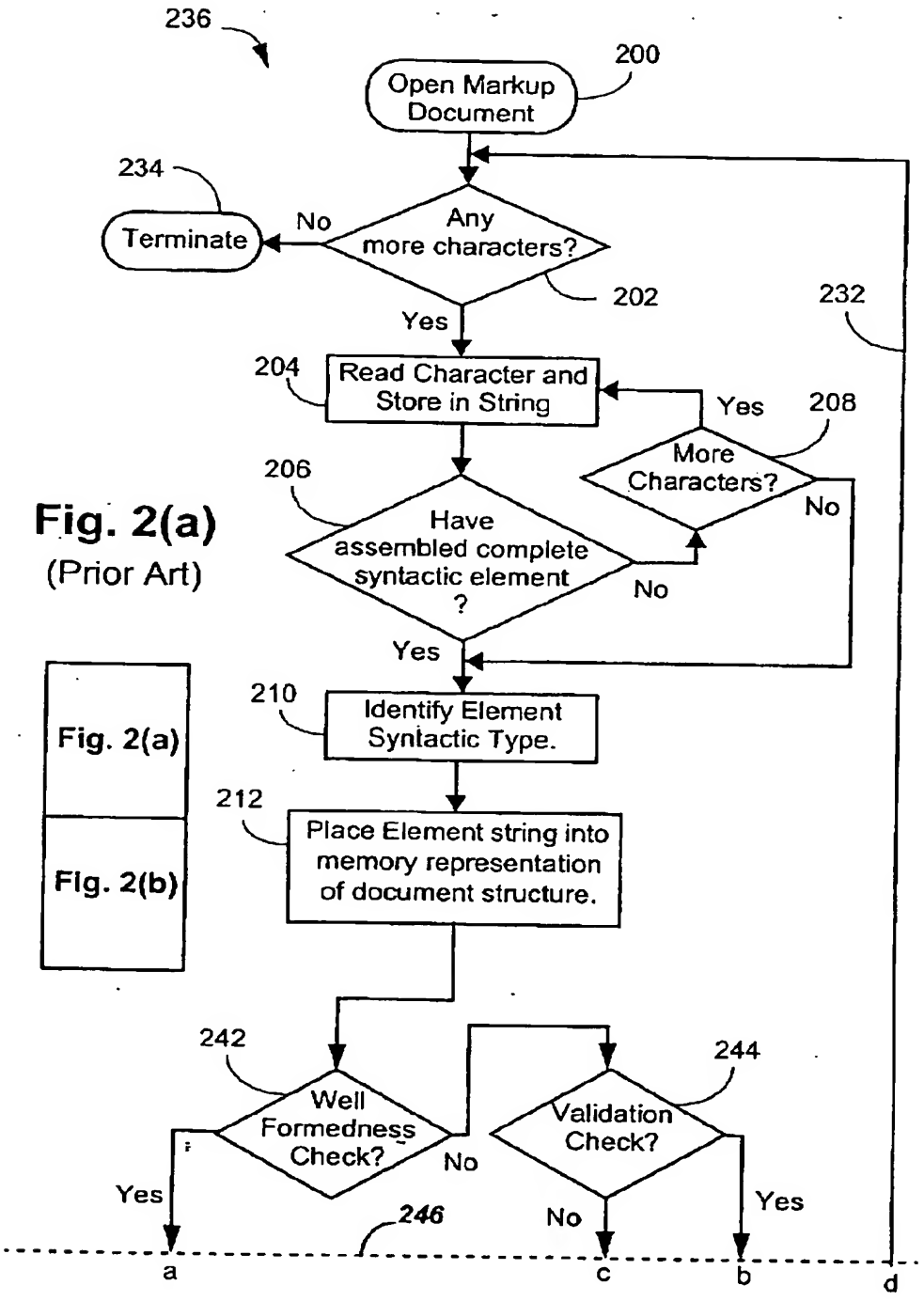


Fig. 2(a)
(Prior Art)

Fig. 2(a)

Fig. 2(b)

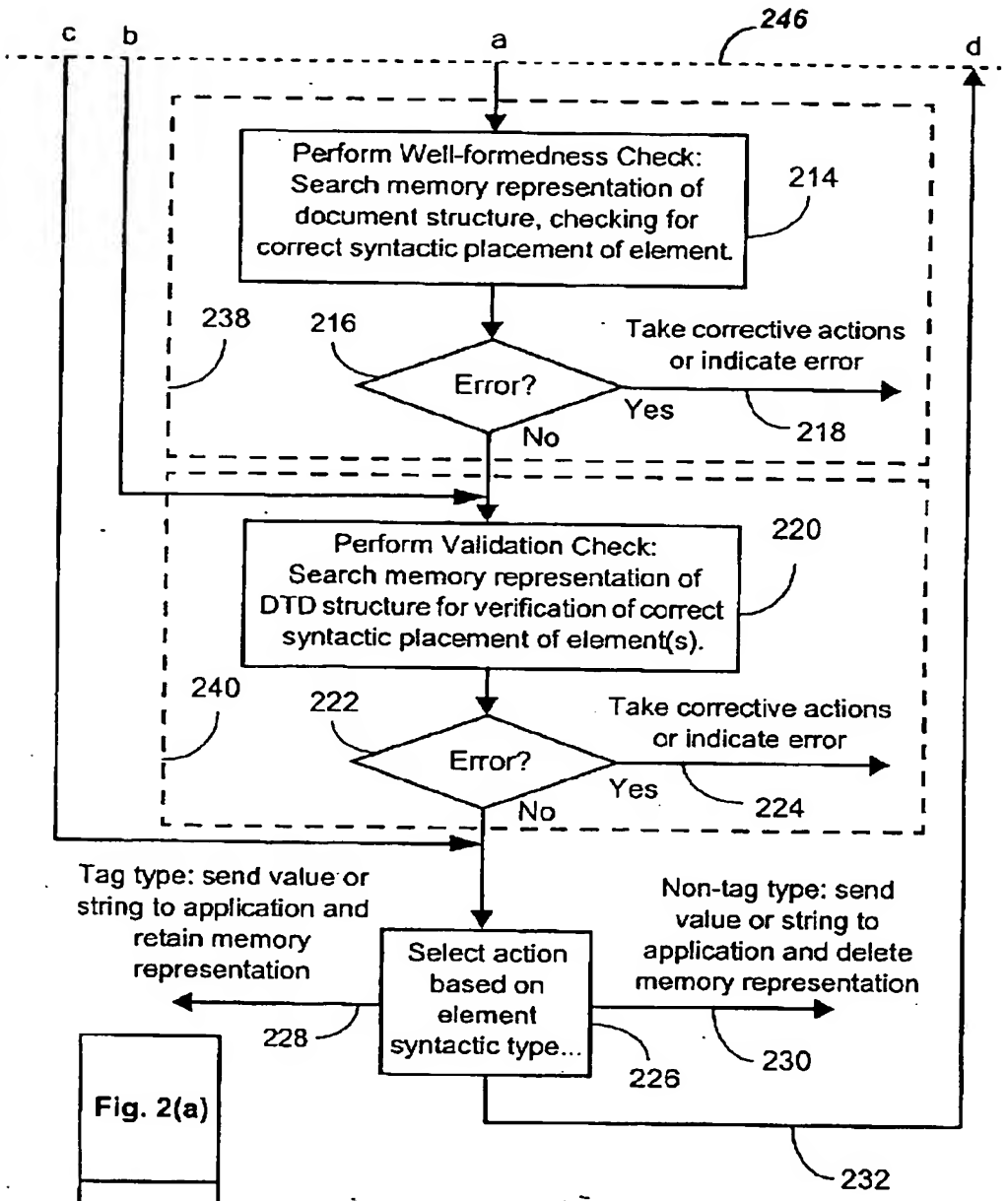


Fig. 2(a)

Fig. 2(b)

Fig. 2(b)
(Prior Art)

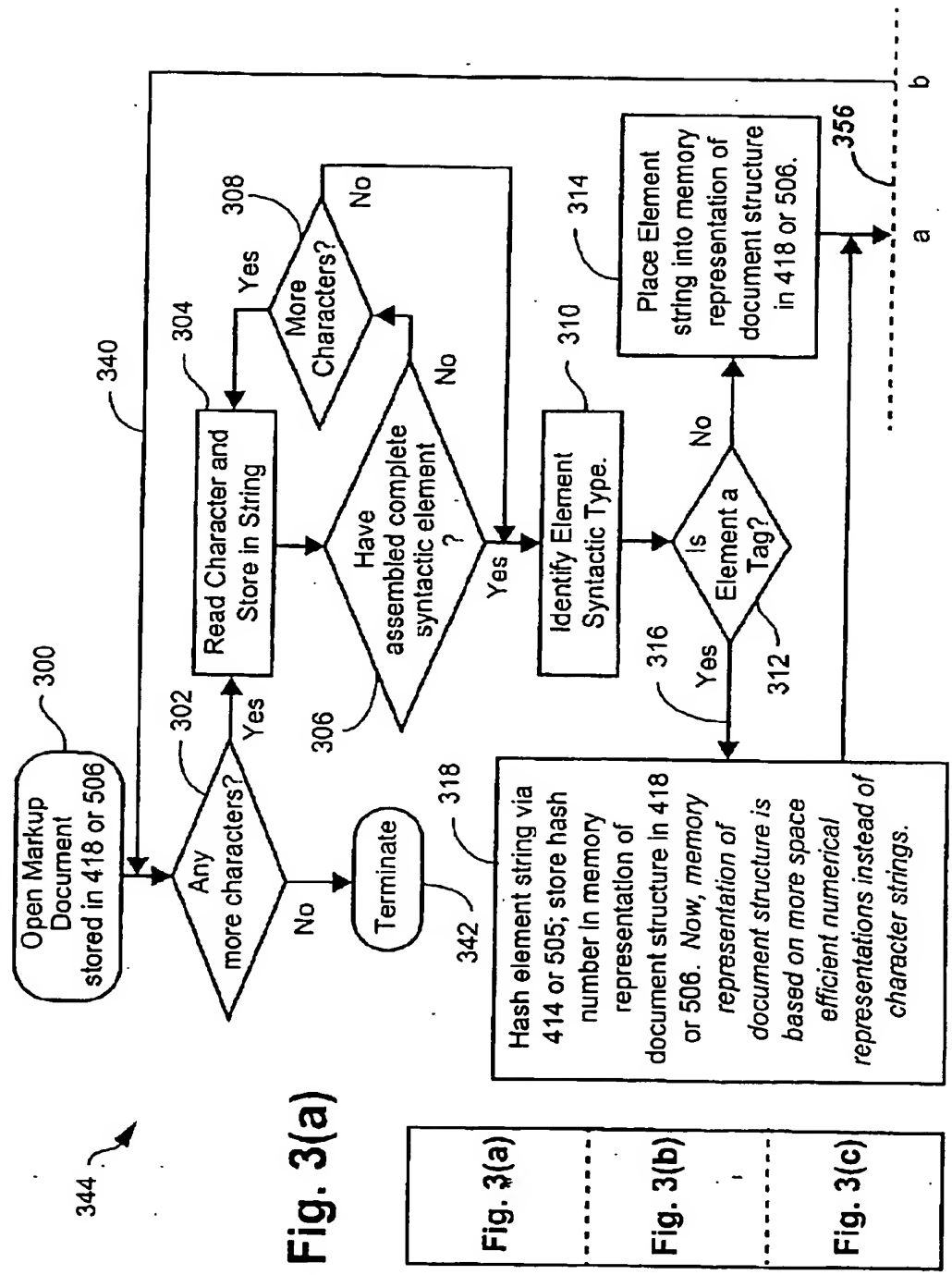

344
Open Markup Document stored in 418 or 506
300
Any more characters?
302
No
Yes
Terminate
318
342
Read Character and Store in String
304
340
More Characters?
308
Yes
No
Have assembled complete syntactic element ?
306
No
Yes
Identify Element Syntactic Type.
310
Is Element a Tag?
312
No
Yes
316
Place Element string into memory representation of document structure in 418 or 506.
314
356
a
b
Hash element string via 414 or 505; store hash number in memory representation of document structure in 418 or 506. Now, memory representation of document structure is based on more space efficient numerical representations instead of character strings.
Fig. 3(a)
Fig. 3(b)
Fig. 3(c)

Fig. 3(a)

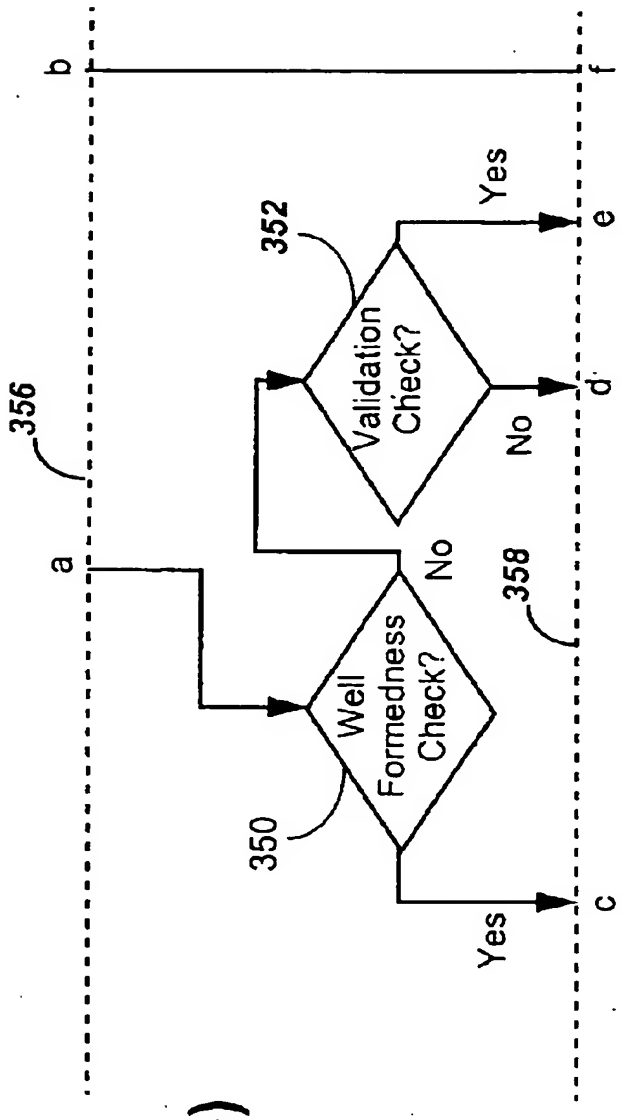

b
f
Yes
352
e
Validation Check?
356
d
No
a
No
358
Well Formedness Check?
350
Yes
c

Fig. 3(b)

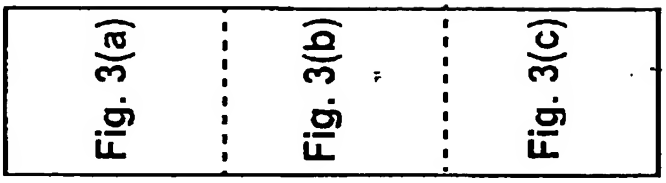


Fig. 3(a)
Fig. 3(b)
Fig. 3(c)

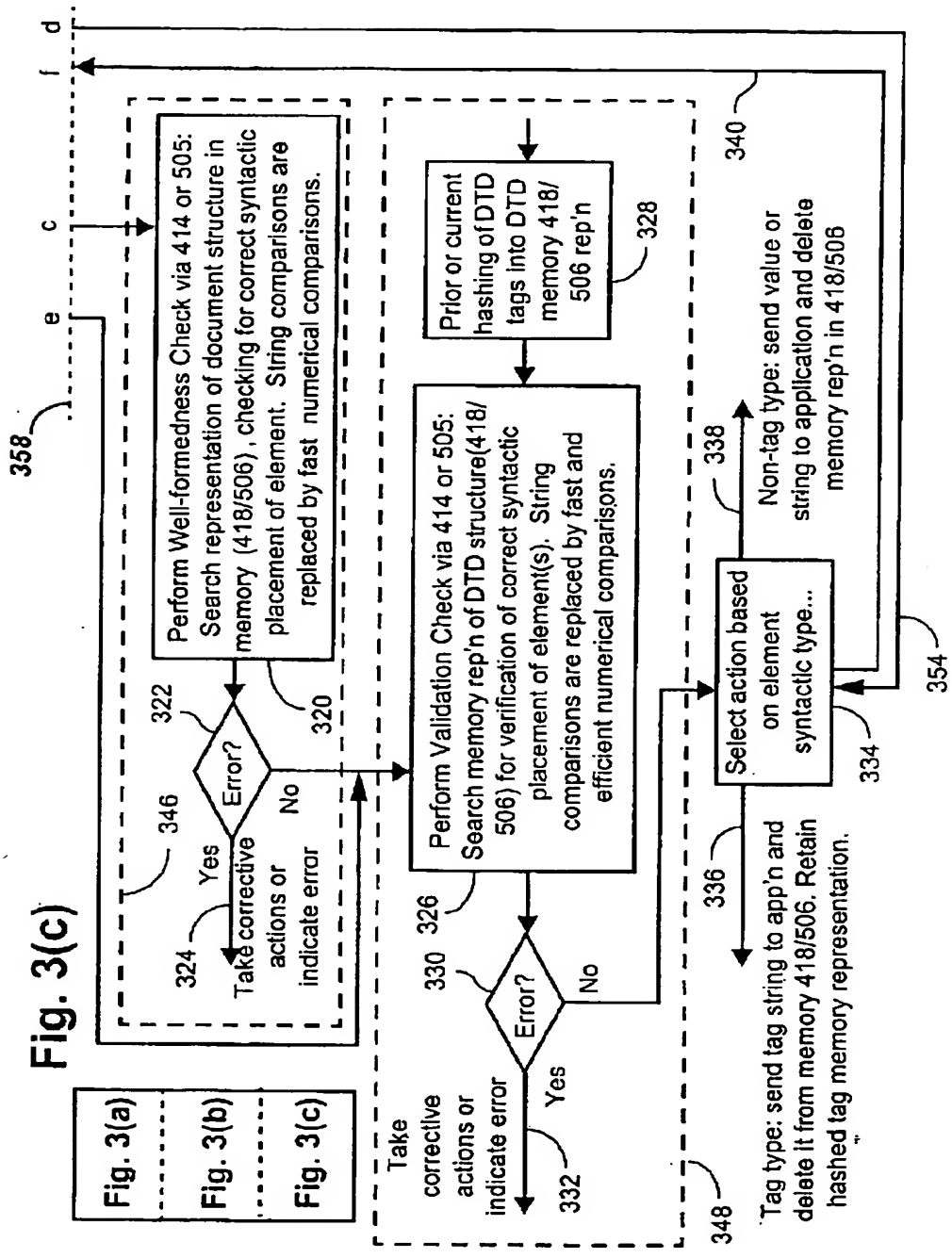


Fig. 3(c)
Fig. 3(a)
Fig. 3(b)
Fig. 3(c)
358
e
c
f
d
Perform Well-formedness Check via 414 or 505: Search representation of document structure in memory (418/506), checking for correct syntactic placement of element. String comparisons are replaced by fast numerical comparisons.
320
322
Error?
Yes
No
346
324
Take corrective actions or indicate error
Perform Validation Check via 414 or 505: Search memory rep'n of DTD structure(418/506) for verification of correct syntactic placement of element(s). String comparisons are replaced by fast and efficient numerical comparisons.
326
328
Prior or current hashing of DTD tags into DTD memory 418/506 rep'n
330
Error?
Yes
No
332
Take corrective actions or indicate error
348
336
Select action based on element syntactic type...
338
Non-tag type: send value or string to application and delete memory rep'n in 418/506
340
334
354
Tag type: send tag string to app'n and delete it from memory 418/506. Retain hashed tag memory representation.

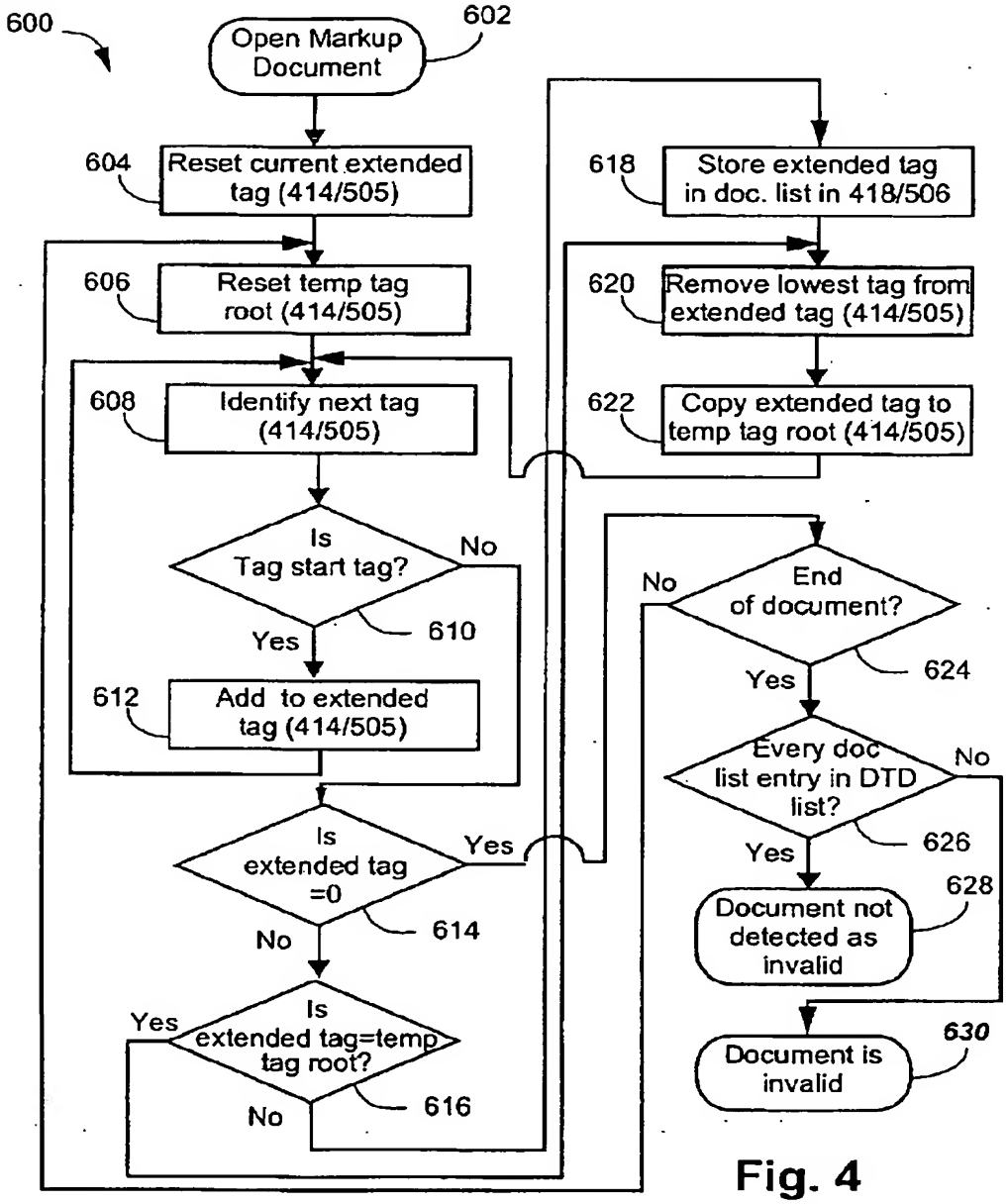

600
Open Markup Document
602
604
Reset current extended tag (414/505)
606
Reset temp tag root (414/505)
608
Identify next tag (414/505)
Is Tag start tag?
No
610
Yes
612
Add to extended tag (414/505)
Is extended tag =0
Yes
614
No
Yes
Is extended tag=temp tag root?
616
No
618
Store extended tag in doc. list in 418/506
620
Remove lowest tag from extended tag (414/505)
622
Copy extended tag to temp tag root (414/505)
No
End of document?
624
Yes
Every doc list entry in DTD list?
No
626
Yes
628
Document not detected as invalid
630
Document is invalid

Fig. 4

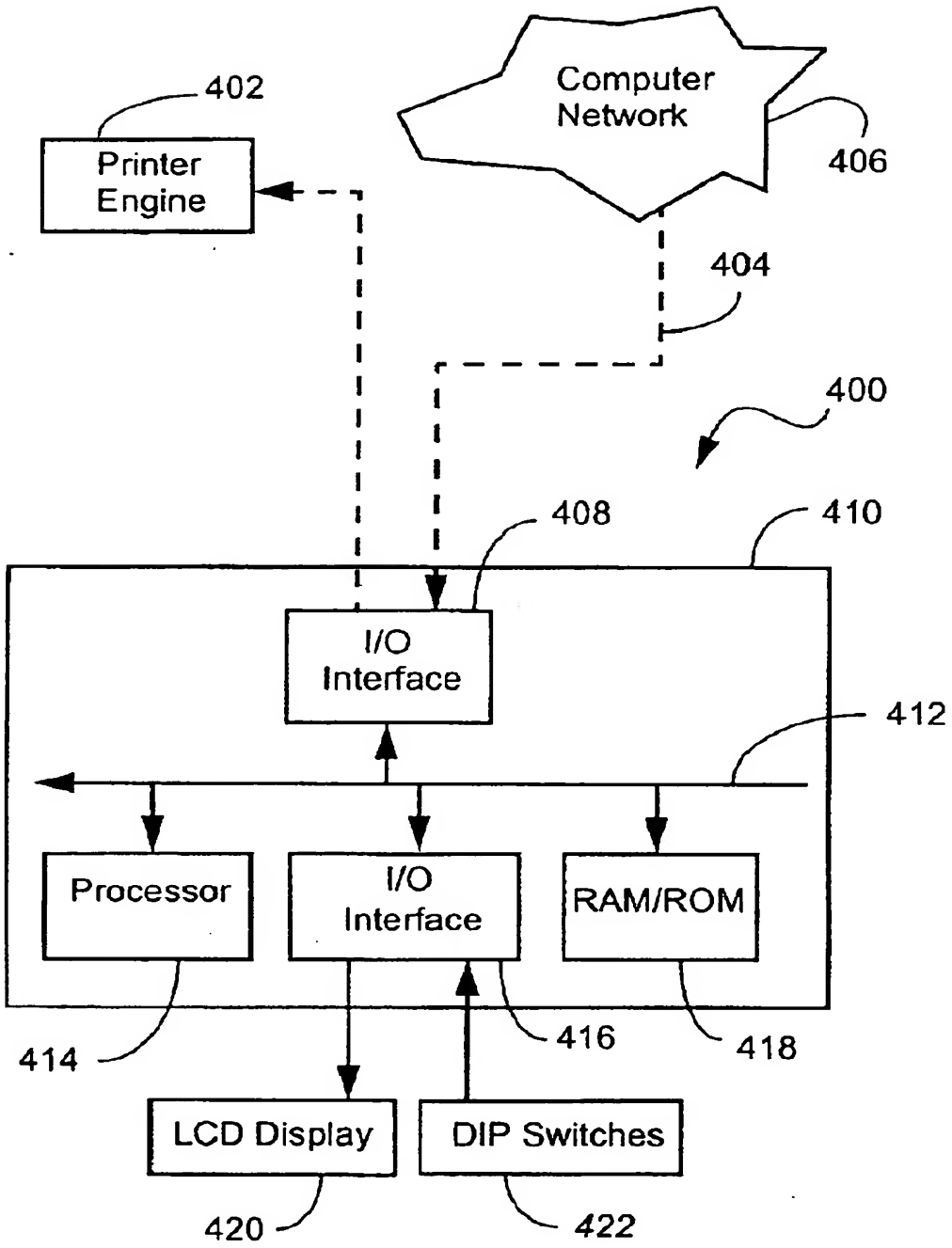

Computer Network
406
402
Printer Engine
404
400
408
410
I/O Interface
412
Processor
I/O Interface
RAM/ROM
414
416
418
LCD Display
DIP Switches
420
422

Fig. 5

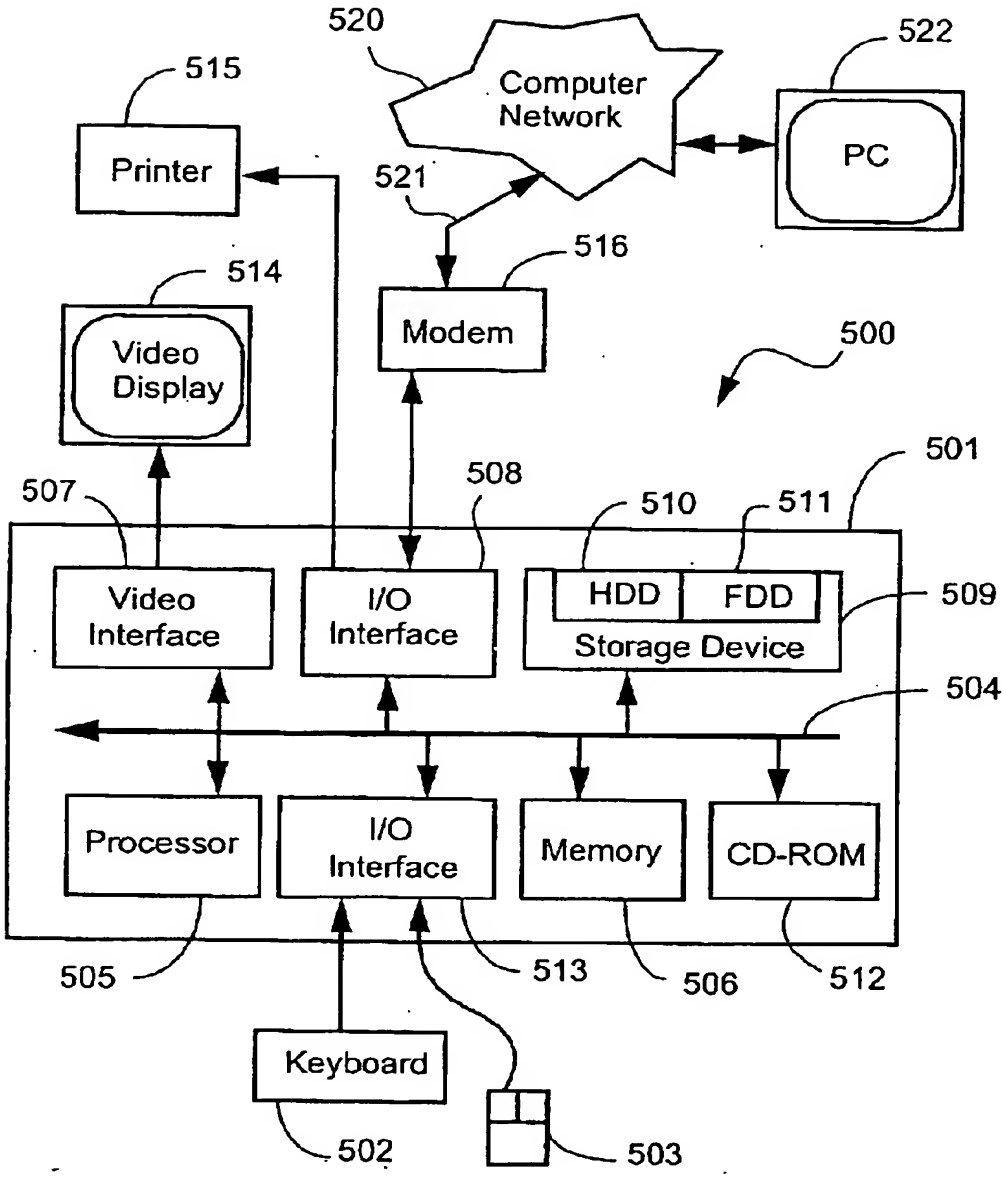

520
Computer Network
522
PC
515
Printer
521
516
Modem
514
Video Display
500
501
507
508
510
511
Video Interface
I/O Interface
HDD
FDD
Storage Device
509
504
Processor
I/O Interface
Memory
CD-ROM
505
513
506
512
Keyboard
502
503

Fig. 6

1. Abstract

A method of parsing a markup language document comprising syntactic elements is disclosed, said method comprising, for one of said syntactic elements, the steps of identifying (310) a type of the element, processing (318) the element by determining a hash representation thereof if said type is a first type, and augmenting (314) an at least partial structural representation of the document using the hash representation if said type is said first type.

2. Representative Drawing

Fig.3(a)